Computer Praxis im Unterricht

Albrecht/Mödl

Steuern und Regeln

mit LEGO Lines und dem LEGO TC Controller

J.B.Metzler | B.G.Teubner

ɔmputerPraxis im Unterricht

Metzler + Teubner Buch- und Diskettenreihe für die
ɜmeine und berufliche Lehrer- und Erwachsenenbildung

ɔrs/Huth: **Computereinsatz im Wirtschaftsunterricht**
256 Seiten. Buch mit Diskette (IBM+kompatible) DM 48,–

'echt/Mödl: **Steuern und Regeln mit LEGO Lines und dem LEGO TC Controller**
208 Seiten. Buch mit Diskette (IBM+kompatible) DM 48,–

mann: **Computereinsatz in Sozialkunde, Geographie und Ökologie**
212 Seiten. Buch mit Diskette (IBM+kompatible) DM 48,–

m/Ehrhardt/Hole: **Schüler arbeiten mit einem Tabellenkalkulationsprogramm**
283 Seiten. Buch mit Diskette (IBM+kompatible) DM 48,–

schhauer/Schindler: **Schüler führen ein Bankkonto**
288 Seiten. Buch mit Diskette (IBM+kampatible) DM 48,–

ɪze/Menzel: **AppleWorks-Praxis**
207 Seiten. Buch mit Diskette (Apple) DM 48,–

ɪze/Menzel/Mödl/Plieninger: **FRAMEWORK-Praxis**
Band 1: Konzepte
254 Seiten. Buch mit Diskette (IBM+kompatible) DM 48,–
Band 2: Anwendungen
272 Seiten. Buch mit Diskette (IBM+kompatible) DM 48,–

rmann/Schmälzle: **Daten und Energie**
224 Seiten. DM 28,80

ɪerich/Schindler/Steigerwald: **Schüler schreiben eine Computer-Zeitung**
288 Seiten. Buch mit Diskette (IBM+kompatible) DM 48,–

ɔerich/Steigerwald: **Schüler arbeiten mit einer Datenbank**
272 Seiten. Buch mit Diskette (IBM+kompatible) DM 48,–

ɪgen/Otto: **Computereinsatz im Unterricht**
260 Seiten. DM 28,80

ß: **Computereinsatz im Erdkundeunterricht**
187 Seiten. Buch mit Diskette (C 64/C 128) DM 48,–

chwitz/Wedekind: **Computereinsatz im Biologieunterricht**
In Vorbereitung

ɪmann/Madlncea/Pannek: **Materialien zur ITG**
Band 1: Unterrichtseinheiten
306 Seiten. Buch mit Diskette (IBM+kompatible) DM 48,–
Band 2: Didaktisch-methodische Hinweise
77 Seiten. DM 14,80

ɪschner: **Computereinsatz im kommunikativen Fremdsprachenunterricht**
In Vorbereitung

nzel/Probst/Werner: **Computereinsatz im Mathematikunterricht**
Materialien für die Klassenstufen 9 und 10

ɪzel/Thode/Plieninger: **Computer-Werkzeug für alle Lehrer –**
Leitfaden für Einsteiger und Umsteiger
199 Seiten. DM 16,80

ɪsetzung auf der 3. Umschlagseite

ComputerPraxis im Unterricht

Herausgegeben von
Rüdeger Baumann, Celle, Dr. Karl-August Keil, Augsburg
Dr. Leo H. Klingen, Bonn, Dr. Klaus Menzel, Schwäbisch Gmünd
Reinhold Thode, Rendsburg

Steuern und Regeln

mit LEGO Lines und dem LEGO TC Controller

Von Helmut Albrecht, Realschullehrer, Heidenheim
und Prof. Dr. Herbert Mödl, Schwäbisch Gmünd

1990

J. B. Metzler Stuttgart

B. G. Teubner Stuttgart

Trademarks

	Geschützte Warenzeichen der Firma
Commodore	Commodore Büromaschinen GmbH
IBM	International Business Machines Corp.
LEGO	LEGO GmbH
MS-DOS	Microsoft Corporation
Turbo C	Borland International, Inc
Turbo Pascal	Borland International, Inc
UniComal	Unicomal A/S
Framework	Ashton-Tate Corp.
WordPerfect	WordPerfect Corp.
HP Laserjet II	Hewlett-Packard Company

Die Konzeption des Buches wurde mit dem Softwarepaket Framework erstellt. Die Feinbearbeitung und Endredaktion erfolgte schließlich in WordPerfect. Der Ausdruck wurde auf einem HP Laserjet Serie II mit den Fonts Times Roman und Letter Gothic vorgenommen.

CIP-Titelaufnahme der Deutschen Bibliothek

Albrecht, Helmut:
Steuern und Regeln mit LEGO Lines und dem LEGO TC Controller / von Helmut Albrecht u. Herbert Mödl. – Stuttgart : Metzler ; Stuttgart : Teubner, 1990
(Computer-Praxis im Unterricht)
ISBN 3-476-70214-6 (Metzler)
ISBN 3-519-02595-7 (Teubner)
NE: Mödl, Herbert:

Printed in Germany
Gesamtherstellung: Präzis-Druck GmbH, Karlsruhe

Vorwort

Als Lehrender in der Aus- und Weiterbildung von Lehrerinnen und Lehrern ist man schon von Berufs wegen ständig aufgerufen und bemüht, neben der Vermittlung bewährten Wissens und der Weitergabe von Erfahrungen die Lehre und den Unterricht mit neuen Inhalten und Medien zu erproben und lebendig zu erhalten. Dies gilt erst recht, wenn diese neuen Inhalte und Medien sogar vom Lehrplan gefordert werden.

So sind beispielsweise mit neuen Lehrplänen für die Haupt- und Realschulen in Baden-Württemberg Mitte der 80er Jahre auf die Schulen Forderungen zugekommen, der Thematik Computer in unserer Arbeitswelt verstärkt Rechnung zu tragen, soweit es die sächlichen und personellen Ausstattungen ermöglichen. Inhaltlich wie methodisch wurde diese Forderung aber weitgehend auf unbearbeitetem Boden abgesetzt.

Bei der Sichtung und Erprobung der in der Literatur und dann aber auch in den von der Lehrmittelindustrie angebotenen Konzepten für diesen neuen Unterricht stießen wir unter anderem auf die Materialien der Firma LEGO. Sie schienen uns wert, in der didaktischen Forschung und Lehre an der Pädagogischen Hochschule genauer untersucht zu werden und nach den ersten Erfahrungen entschlossen wir uns, diese in Buchform einer breiteren Leserschaft zugänglich zu machen.

Als wir die Notwendigkeit verspürten, mit LEGO Lines auch in eine Schule zu gehen und die Materialien mit dem Adressatenkreis, für den sie gedacht sind, den Schülerinnen und Schülern, praktisch zu erproben, hatten wir großes Glück. Es gab nämlich im weiteren Umkreis unserer Pädagogischen Hochschule nur eine einzige Schule, die eine dafür geeignete Computerausstattung besaß. Die Software LEGO Lines ist nämlich nur für zwei Computertypen verfügbar und der MS-DOS-Computer als Standard beginnt erst jetzt, die Homecomputerausstattung in den Schulen abzulösen bzw. zu ergänzen.

An der Adalbert-Stifter-Realschule und auf dem Schulamt in Schwäbisch Gmünd begegnete man unserem Anliegen mit Interesse und Wohlwollen. Ein gemeinsamer Unterrichtsversuch im Rahmen einer Arbeitsgemeinschaft bot die Möglichkeit, eine trotz Lehrplanforderung bislang an dieser Schule noch nicht verfolgte neue Qualität in den unterrichtlichen Einsatz des Computers zu bringen.

Die unterrichtliche Erprobung bereitete dann allen Beteiligten, den Lehrenden wie vor allem auch den Schülern, viel Spaß, und wir dürfen behaupten, daß wir daraus alle, jeder für seinen Teil, brauchbare Einsichten gewonnen haben. Leider waren die Teilnehmer an dieser Arbeitsgemeinschaft nur Jungen. Verstehen Sie darum bei der Lektüre des Buches, wenn immer nur von Schülern und Lehrern die Rede ist,

unsere Diktion auch vor diesem Hintergrund. Natürlich meinen wir mit diesen Redewendungen auch immer die Mädchen und die Lehrerinnen.

Noch ist es uns aber leider nicht gelungen, für diese mehr technisch orientierte Computernutzung auch unsere Frauenwelt zu motivieren. Wir haben jedoch die Hoffnung, daß gerade Materialien wie die von LEGO geeignet sind, auch Mädchen für diese Art der Computeranwendung, die uns in unserer täglichen Lebenswelt auf Schritt und Tritt begegnet, zu interessieren.

Die mächtige Erweiterung der Computersteuerung von LEGO Modellen durch den LEGO TC Controller ließ es uns sinnvoll erscheinen, im zweiten Teil des Buches auch noch auf dieses Produkt einzugehen und es vorzustellen, auch wenn wir hier noch keine eigenen Unterrichtserfahrungen einbringen können.

Unser Buch richtet sich vornehmlich an Kolleginnen und Kollegen in den Schulen. Es will Entscheidungshilfe sein und Anregungen vermitteln. Wir danken an dieser Stelle den Kollegen der obengenannten Realschule, den Herren RL Manfred Häfner und RL Georg Hruschka von Hochstamm für die kooperative Zusammenarbeit, aus der wir viele Anregungen für unser Buch verwerten konnten. Wir können uns aber auch vorstellen, daß unser Buch ebenso Eltern und Jugendlichen eine wertvolle Hilfe sein kann, um auf diesem Weg zu Erfahrungen mit computergesteuerten Lernmaterialien voranzukommen.

Die Firma LEGO unterstützte unser Vorhaben dankenswerterweise mit der Überlassung der dazu nötigen LEGO Hardware. Für den Etat vieler Schulen gibt es bei der Anschaffung von Klassensätzen leider noch den Unterschied zwischen preiswert und erschwinglich. Dieses Buch kann vielleicht mithelfen, das Ziel der zu überspringenden Latte für manche erstrebenswerter und weniger hoch erscheinen zu lassen. Als weitere Hilfe dazu sei auch auf das unter maßgebender Mitwirkung von Herrn RL Viktor Wagner bei J. B. Metzler und B. G. Teubner erschienene Videoband "Steuern und Regeln mit LEGO Lines" verwiesen.

Wir danken den Verlagen J. B. Metzler und B. G. Teubner für die entgegenkommende Unterstützung unseres Vorhabens und dem Mitherausgeber dieser CPU-Reihe, dem Kollegen Prof. Menzel, für seine hilfreichen Anregungen und Ermunterungen.

Schwäbisch Gmünd, im Winter 1989/90

Helmut Albrecht Herbert Mödl

Inhaltsverzeichnis

1 Einführung

Computergesteuerte Werkzeugmaschinen und rechnerunterstützte Fertigungsstraßen sind aus den heutigen Arbeitsprozessen nicht mehr wegzudenken. Es ist deshalb konsequent, die Fähigkeit des Computers zur Steuerung von Maschinen bereits im Unterricht aufzuzeigen. Der Bildungsplan der Hauptschule des Landes Baden-Württemberg nennt beispielsweise im Fach Technik die "Bedienung, Überwachung und Kontrolle von Arbeitsmaschinen durch computergesteuerte Rechner" als Unterrichtsinhalt in Klasse 8. Die Schulen sind damit vor die Aufgabe gestellt, neben den hierzu nötigen Computern auch noch entsprechende Funktionsmodelle vorzuhalten.

Der von den Lehrplänen in diesem technischen Bereich geforderte Unterricht stellt insbesondere an die Lehrer enorme Anforderungen, die diese Forderungen letztendlich verantwortlich umzusetzen haben. Sie müssen dazu mit dem Medium Computer sowohl von Seiten der Software wie von Seiten der Hardware kompetent umgehen können. Reicht es für die überwiegende Anzahl der normalen Computeranwender doch noch aus, wenn sie die von der Software an sie gestellten Anforderungen meistern können. Keineswegs Standard ist es jedoch, daß Computernutzer sich an Aufgaben heranzuwagen haben, die dem Bereich der Hardware zugerechnet werden müssen. Schon der Anschluß eines Druckers bereitet ja vielen Computeranwendern große Schwierigkeiten. Darum sind gerade für die Lehrerschaft auch für diese ihre Aufgabenerfüllung genügend geeignete Unterrichtsmaterialien und brauchbare Hilfen bereitzustellen.

Inzwischen sind von einzelnen Lehrern oder Lehrergruppen Modelle entworfen und zum Nachbau veröffentlicht, bzw. als Bausätze im Angebot. Hierbei handelt es sich zumeist um funktionsspezifische Anlagen: einfache pneumatisch betriebene Roboter, Bohranlagen, Styroporschneider, Zahlenschlösser, Kilometerzähler und dergleichen mehr. Mit dem Terminus "funktionsspezifisch" soll hier ausgedrückt werden, daß die konzipierten und gebauten Modelle nur eine einzige bestimmte Funktion ausführen können.

Als Alternative dazu bietet es sich an, vielfältige und variable Funktionsmodelle, wie sie mit technischen Baukästen herstellbar sind, im Unterricht zu planen, sie aufzubauen und mit dem Computer zu steuern. Die

Vielzahl und Vielfalt der realisierbaren Modelle sowie die für Schüler motivierende aktive Hinführung zu aktuellen technischen Problemstellungen sprechen für diesen Lösungsweg. Neben der Variabilität dieses Weges sind hier zudem die in der Regel mitgelieferten Softwarehilfen zum Ansteuern der Modelle hervorzuheben. Auf die Einwände gegen den unterrichtlichen Einsatz von Modellbaukästen gibt es gerade im Bereich der informationstechnischen Grundbildung, wozu ja auch der in diesem Buch behandelte Bereich zählt, vielfältige Antworten, die wir dann konkret bei der Darstellung der verschiedenen Projekte in Kapitel 4 geben werden.

Der ansonsten in der Schule übliche Weg, die gebauten Modelle und Anlagen mit den PEEK- und POKE-, bzw. INP- und OUT-Befehlen des jeweiligen BASIC-Dialektes des verwendeten Computers zu steuern, ist denkbar unbefriedigend. Wir denken hierbei weniger an die allgemein bekannten Schwächen der nur sehr gering strukturierten Programmiersprache BASIC, die zu unsauberem Programmieren verleiten und zu unübersichtlichen und damit schwer nachvollziehbaren Listings führen. Es soll hier bestimmt keine fundierte Programmierausbildung im Sinne eines heimlichen Informatikunterrichts an den Schulen propagiert werden; andererseits ist es aber auch sicher unstrittig, daß alle Schüler die im Unterricht erarbeiteten Ergebnisse nachvollziehen können sollten. Gerade dies scheint jedoch mit Blick auf die hierbei häufig produzierten BASIC-Listings, die zumeist keine Strukturierung erkennen lassen, mehr als fraglich zu sein.

Das Hauptargument gegen den Einsatz der Programmiersprache BASIC zu Steuerzwecken in der Schule liegt aber vielmehr beim hierzu notwendigen Verständnis des Dualsystems. Wie noch gezeigt werden wird, sind sämtliche Daten im Computer im Dualsystem codiert. Bei allen übrigen Anwendungen des Computers bleibt diese binäre Informationscodierung dem Benutzer verborgen. Lediglich bei Steuerungsaufgaben müssen die Wertigkeiten der einzelnen Steuerleitungen bestimmt werden, wofür Umrechnungen zwischen dem bekannten Dezimal- und dem häufig unbekannten Dualsystem notwendig werden. Wie zäh solche Umrechnungen selbst Erwachsenen von der Hand gehen, wird immer wieder in entsprechenden Seminaren und Fortbildungen deutlich. Ein Schüler mag zwar auswendiglernen können, daß beispielsweise der Dezimalwert 16 an seiner Modellampel die grüne Lampe leuchten läßt, daß 8 gelb und 4 rot bedeutet und er mit dem Wert 12 die gelbe und rote Lampe leuchten

lassen kann, verstehen wird er den Zusammenhang in der Regel jedoch nicht. Fordern nicht aber unsere Lehrpläne gerade das Verständnis von elementaren Vorgängen und Einsichten in grundlegende Zusammenhänge?

Ein dritter Nachteil bei der Verwendung der Programmiersprache BASIC ist die dabei notwendige genaue Hardwarekenntnis. Jeder Computertyp organisiert seinen Ein- und Ausgabebereich völlig anders. Für den verwendeten Typ sind Kenntnisse über die Art und Anzahl der vorhandenen Ein-/Ausgabeports sowie deren Adressen notwendig. All dieses Wissen wird dann beim Wechsel des Computertyps völlig wertlos und man wird sich wieder total neu einarbeiten müssen. Für den Schüler stellt solches Wissen deshalb nur unnötigen Ballast dar, den er mit sich herumschleppen muß.

Noch gar nicht genannt wurden in diesem Zusammenhang das notwendige Erlernen der Programmiersprache BASIC, die für viele Schüler (und auch für Lehrer) ja ebenfalls neu sein dürfte und das Einarbeiten in das verwendete Betriebssystem, um Dateien und Programme laden und abspeichern zu können. Bevor die erste Lampe an der Modellampel brennt oder sich der Arm des angeschlossenen Modellroboters zum ersten Mal bewegt, sind somit auf dem "traditionellen" Weg sehr viele Hindernisse zu beseitigen.

Diese Schwierigkeiten versuchen die Hersteller von Konstruktionsbaukästen durch mitgelieferte Software zu beseitigen. Hierbei ging man zunächst den Weg einer Befehlserweiterung der Programmiersprache BASIC. Es wurden Maschinenprogramme dazugeladen, die es dem BASIC-Interpreter ermöglichten, auch Befehle, wie beispielsweise MOTOR(1,EIN) usw. zu "verstehen", und entsprechende Reaktionen zu initiieren. Damit war bereits die spezifische Hardwarekenntnis und ein Rechnen im Dualsystem überflüssig geworden. Die Programmiersprache BASIC mit ihren Unzulänglichkeiten und die nach wie vor notwendige Einarbeitung in das Betriebssystem waren jedoch geblieben.

Seit 1987 bietet die Firma LEGO im Zusammenhang mit ihrem Technic-System die Steuerungssoftware LEGO Lines an, die alle zuvor genannten Nachteile und Schwierigkeiten vermeidet. Geführt von einer graphischen Benutzeroberfläche kann der Schüler direkt und ohne Umrechnungen die Schaltzustände der Ausgabeleitungen verändern und auf einen Blick die

Signalpegel der Eingabeleitungen ablesen. Die wenigen notwendigen "Befehle" sind in deutscher Sprache eindeutig in sogenannten Strukturanweisungen formuliert, die auch tatsächlich zu einer Strukturierung des Programms beitragen. Die notwendigen Betriebssystemfunktionen wie Speichern, Laden und Löschen von Steuerungsprogrammen werden über klar gegliederte und aussagekräftige Menüs ausgeführt. Ein geeigneter Dialog hilft Fehlentscheidungen zu vermeiden und falsch getroffene Auswahlen zurückzunehmen. Schüler können bereits nach kürzester Zeit selbständig mit LEGO Lines arbeiten. Neben der damit verbundenen hohen Motivation ist als weiterer positiver Effekt die Entlastung des Lehrers zu nennen, der nun nicht mehr ständig mit Erklärungen, Überwachung und Fehlersuche beschäftigt ist. Für diese hervorragende Konzeption erhielt LEGO Lines 1987 den Schul-Software-Preis.

LEGO Lines ist für schulische Anforderungen bis einschließlich Sekundarstufe I völlig ausreichend. Für komplexere Steuerungsaufgaben oder für eine anspruchsvolle Behandlung der Prozeßsteuerung in der Sekundarstufe II bietet LEGO seit 1989 den LEGO TC Controller an. Neben einer leistungsfähigen Sprachergänzung zu Turbo Pascal bietet der Controller eine einzigartige Möglichkeit der Direktsteuerung angeschlossener Modelle über die Computertastatur. Damit können auf schnelle und einfache Weise Startpositionen angefahren, Eingangszustände abgefragt und Leistungspegel eingestellt werden. Durch die Schnittstelle zu Turbo Pascal werden alle Strukturen und Möglichkeiten dieser Programmiersprache für Prozeßsteuerungen zur Verfügung gestellt.

2 Steuern mit dem Computer

Um mit dem Computer irgendwelche Steuerungsaufgaben zu bewältigen, ist ein Datenaustausch zwischen Computer und Umwelt nötig. Für diesen Datenaustausch müssen Signale aus dem Computer heraus und auch in ihn hinein gelangen können. Da jeder Computer für einen ordnungsgemäßen Betrieb an die Steckdose angeschlossen werden muß, liegt die Vermutung nahe, daß es sich hierbei um elektrische Signale handelt. Steckbuchsen und Kontaktleisten finden sich auf der Rückseite eines jeden Rechners, so daß die angestellten Überlegungen leicht mit einem Voltmeter überprüft werden können. Selbst bei einem unsystematischen Messen wird dabei auffallen, daß die gefundenen Meßwerte entweder rund 0 Volt oder aber knapp 5 Volt betragen.

Die zur Informationsdarstellung im Computer verwendeten Spannungssignale können keine beliebigen gleitenden (analogen) Werte annehmen, sondern lediglich die gemessenen Werte von 5 und 0 Volt. Diese beiden Spannungs- und Informationszustände werden als HIGH und LOW bezeichnet. Da durch Leitungsverluste nicht immer genau die angegebenen Spannungswerte garantiert werden können, wurden Bereiche festgelegt, innerhalb deren Grenzen ein Spannungssignal als HIGH- oder als LOW-Pegel gewertet wird. Sind die im Computer verwendeten integrierten Logikbausteine aus Transistoren aufgebaut sind, spricht man auch von einer Transistor-Transistor-Logik oder kurz von TTL-Bausteinen.

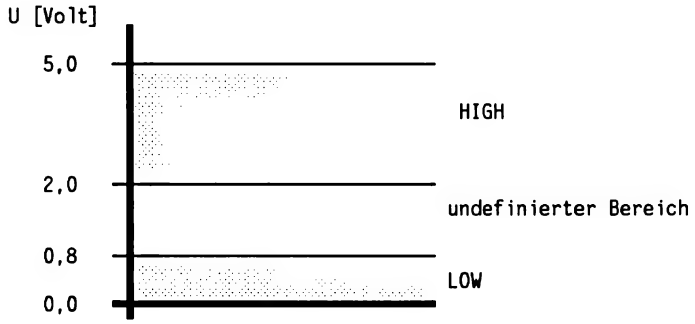


Abb. 2.1 TTL-Pegel

Arbeitet man in der Digitalelektronik mit zwei klar unterscheidbaren Signalzuständen, wie hier mit HIGH und LOW, so spricht man von der **Binärtechnik**. Eine Datenleitung im Computer kann dabei nur zwei verschiedene Informationen (0 bzw. 5 Volt) transportieren. Ein solches binäres Signal wird als "binary digit" oder abgekürzt "**Bit**" bezeichnet. Natürlich wären einzelne Bits für den Informationsumsatz im Computer viel zu wenig, weshalb man mehrere Bits in Gruppen zusammenfaßt. In den meisten Computern ist eine Zusammenfassung von acht Bits üblich, die man ein **Byte** nennt. Mit einem solchen Byte lassen sich nun schon 256 Informationen verschlüsseln.

Denkt man sich die acht Bits eines Bytes in einer festgelegten Ordnung "nebeneinandergelegt" und legt man zudem fest, daß jeder HIGH-Pegel eine 1 und jeder LOW-Pegel eine 0 repräsentiert, so können mit einem Byte die Dualzahlen von 0000 0000 (dezimal: 0) bis 1111 1111 (dezimal: 255) dargestellt werden. Diese naheliegende Interpretation von maschineninternen Signalpegeln als Dualzahlen, die uns Menschen etwas verständlicher sind, brachte es mit sich, daß die Computerpioniere der ersten Stunde im Dualsystem "arbeiten" mußten, was für jemanden, der gewohnt ist, mit seinen zehn Fingern zu rechnen, gar nicht so leicht ist! Auch wir werden deshalb diese fremde Welt zunächst wieder verlassen und uns dem näherliegenden Problem zuwenden, wie denn nun Signale aus dem Computer heraus gelangen können.

Hierzu muß man wissen, daß alle Ein-/Ausgabeeinheiten der Computer unter festgelegten Adressen anzusprechen sind. Der Druckeradapter des IBM PC beispielsweise hat meistens die leicht zu merkende Adresse 888. Befindet sich Ihre Druckerschnittstelle jedoch auf der Herculeskarte, so können Sie sie unter der Adresse 956 ansprechen. Beim Blättern im BASIC-Handbuch findet man den Ausgabebefehl

OUT <adresse>, <wert>

der zur Lösung der gestellten Aufgabe benötigt wird. Im Handbuch ist dazu zu lesen, daß der OUT-Befehl die in <wert> angegebene ganze Zahl an den mit <adresse> spezifizierte Ein-/Ausgabeport sendet. Dieses "Senden" bedeutet dabei gleichzeitig ein "Umwandeln" des angegebenen Dezimalwertes in eine Dualzahl, deren Bitmuster dann am Ausgabeport erscheint. Um damit gezielt einzelne Leitungen ein- und ausschalten zu können, wird man sich dann wohl oder übel doch etwas näher mit den ungewohnten Dualzahlen beschäftigen müssen.

Genau wie unser gewohntes Zehnersystem auf den Zehnerpotenzen 1, 10, 100, ... aufbaut, gründet das Zweiersystem auf den Zweierpotenzen:

128 64 32 16 8 4 2 1

die auch geschrieben werden können als:

2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0

Damit wird klar, warum die Datenleitungen nicht von 1 bis 8, sondern von 0 bis 7 durchnumeriert sind: Die Leitungsnummern entsprechen den Exponenten der Zweierpotenzen, womit sofort eine eindeutige Zuordnung möglich wird.

Hierzu ein Beispiel: Die Datenleitungen des Druckerports sollen mit folgenden Pegeln beschaltet werden:

D7 D6 D5 D4 D3 D2 D1 D0

HIGH LOW LOW **HIGH** LOW **HIGH** **HIGH** LOW

Dies entspricht der Dualzahl:

1 0 0 **1** 0 **1** **1** 0

die umgerechnet wird in die Dezimalzahl:

$$\mathbf{1}\cdot 2^7 + \mathbf{0}\cdot 2^6 + \mathbf{0}\cdot 2^5 + \mathbf{1}\cdot 2^4 + \mathbf{0}\cdot 2^3 + \mathbf{1}\cdot 2^2 + \mathbf{1}\cdot 2^1 + \mathbf{0}\cdot 2^0 = 150$$

Nach der Ausgabe des Befehls

OUT 888,150

liegt das gewünschte Pegelmuster am Druckerport an, was Sie leicht mit einem Voltmeter nachprüfen können.

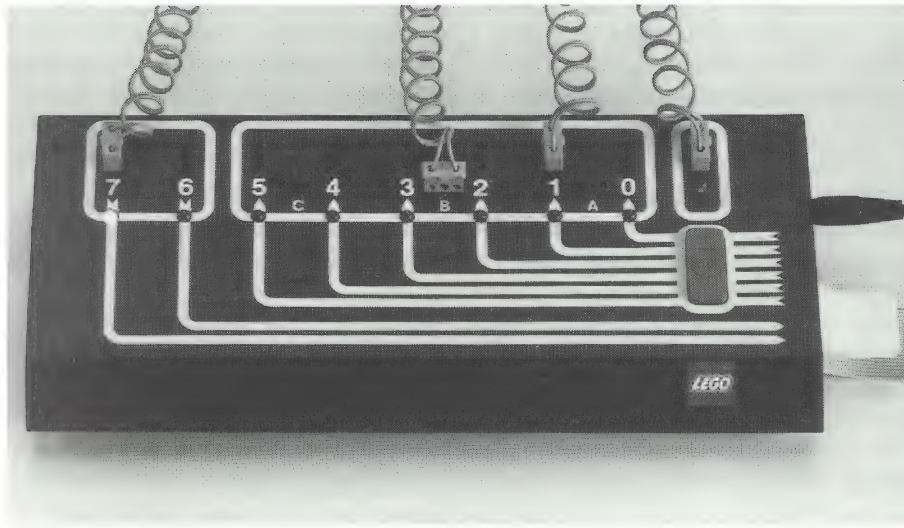


Abb. 3.1 Das LEGO Interface A

3 Das LEGO Technic Control System

3.1 Das LEGO Interface A

Die am Ausgabeport eines Computers anliegenden Steuersignale dürfen keinesfalls direkt zum Betreiben von Lampen oder Elektromotoren verwendet werden. Zwar wird jede aktivierte Ausgabeleitung ein Spannungssignal von etwa 5 Volt führen, der zugehörige Ausgabebaustein im Computer ist jedoch nicht in der Lage, die zum Betreiben der genannten Verbraucher nötige Stromstärke bereitzustellen. Lediglich wenige Milliampere kann eine Ausgabeleitung liefern, gerade ausreichend, um eine Leuchtdiode mit entsprechend dimensioniertem Schutzwiderstand aufleuchten zu lassen. Die elektrische Leistung errechnet sich aus dem Produkt von Spannung und Stromstärke. Die mit den angedeuteten Werten erreichbaren Leistungen reichen niemals aus, um gängige Verbraucher wie Lampen oder gar kleine Elektromotoren zu versorgen. Versucht man es trotzdem, so erzeugt man damit an der entsprechenden Ausgabeleitung praktisch einen Kurzschluß, was der zugehörige Ausgabebaustein im Computer mit seinem Ausscheiden aus dem Dienst quittieren wird. Da die integrierten Schaltkreise im Computer meist direkt auf die Platine gelötet sind, ist die dann notwendige Reparatur eine umständliche und damit kostspielige Angelegenheit.

Die vom Rechner gelieferten Signale müssen somit erst verstärkt werden, bevor sie größere Verbraucher steuern können. Außerdem ist es sehr sinnvoll, den gesteuerten externen Laststromkreis vom Stromkreis des Computers galvanisch zu trennen, um bei Fehlern im Laststromkreis mögliche schädliche Rückwirkungen auf den Rechner mit Sicherheit ausschließen zu können. Diese Aufgaben erledigt das LEGO Interface. Die Computersignale bleiben durch Optokoppler im Interface galvanisch vom Laststromkreis getrennt und werden über schnelle Leistungstransistoren ausreichend verstärkt. Ein separates Netzteil übernimmt die Versorgung dieser Schnittstelle und der daran angeschlossenen Verbraucher. Das LEGO Interface ist völlig verpolungssicher und kurzschlußfest, beides sind wichtige Voraussetzungen für den Einsatz in der Schule!

Die äußere Gestaltung ist ebenfalls schulgerecht: Die Buchsen zum Anschluß von Lampen, Motoren und Eingabesensoren sind in Gruppen zusammengefaßt, die Richtung des Datentransports wird durch aufgedruckte Pfeile angegeben. Die Ausgabeleitungen können, um bei

Fehlprogrammierung eine Beschädigung des Modells zu vermeiden, über einen roten STOP-Schalter schnell abgeschaltet werden.

Über dem STOP-Schalter befindet sich eine abgesetzte Buchse, an welcher ständig 4 Volt Gleichspannung anliegen. Hier dürfen bis zu zwei LEGO Leuchtsteine oder ein Motor angeschlossen werden. Die Leuchtdiode an diesem Permanentausgang leuchtet auf, sobald das Interface über das externe Netzteil mit elektrischer Energie versorgt wird.

Die Ausgabebuchsen sind im nächsten Feld angeordnet. Sechs senkrechte Buchsen sind mit den Ziffern von 0 bis 5 durchnumeriert, drei waagerecht liegende Buchsen mit den ersten drei Buchstaben des Alphabets versehen. Die Ausgabebuchsen können insgesamt bis zu sechs LEGO Leuchtsteine oder drei 4,5 Volt LEGO Motoren versorgen. Der Schaltzustand eines jeden der sechs Ausgabekanäle wird durch eine zugehörige rote Leuchtdiode angezeigt, die beim Aktivieren des jeweiligen Kanals aufleuchtet.

Im linken Feld sind die beiden Signaleingänge zum Anschluß der LEGO Optosensoren oder der LEGO Tastensensoren zusammengefaßt und mit 6 und 7 bezeichnet. Jedem Eingabekanal ist eine grüne Leuchtdiode zugeordnet, die den aktiven oder inaktiven Zustand des angeschlossenen Sensors anzeigt.

Das LEGO Interface wird über ein spezielles Flachbandkabel am Computer angeschlossen. Zum Anschluß am Commodore 64 oder 128 wird das zugehörige Kabel mit dem passenden Stecker direkt am USER-Port eingesteckt. Dieser USER-Port im Commodore läßt sich über die Adresse 56577 ansprechen.

Das Kabelset für den IBM PC und Kompatible enthält zusätzlich eine Adapterkarte zum Einstecken in den Rechner. Damit steht dann im Computer ein spezieller Ein-/Ausgabeport mit der Adresse 925 zur Verfügung. Die Adapterkarte besitzt an ihrer Rückseite eine Buchse zum Anschluß des LEGO Interfaces über das mitgelieferte Flachbandkabel.

3.2 Elemente der LEGO Technic Control Sets

Die für den Aufbau der Modelle nötigen Materialien sind in Systemsets zusammengestellt. LEGO Technic Control I ist als Grundset, LEGO Technic Control II als unabhängiges Erweiterungsset konzipiert. Beide Sets werden mit ausführlichen und detaillierten Anleitungen zum Zusammenbau der vorgeschlagenen Modelle geliefert. Durch die genaue Darstellung aller Bauphasen fällt der Nachbau den Schülern leicht. Übersichtliche Verkabelungsvorschläge machen den Anschluß der Modelle an das Interface sehr einfach.

Mag man auch zunächst diese genauen Bauanleitungen ablehnen, da sie den Schülern keinerlei kreativen Freiraum lassen, so wird ihre Berechtigung sicherlich bereits in den ersten Unterrichtsstunden deutlich. Ohne Verwendung dieser Bauanleitungen werden durch unterschiedliche Schülerfertigkeiten beim Umgang mit dem LEGO Material die benötigten Konstruktionszeiten und -ergebnisse stark voneinander abweichen. Ein gemeinsames Erstellen oder Besprechen der Steuerungsprogramme ist dann erstens durch den unterschiedlichen Baufortschritt und zweitens durch starke Abweichungen in der "Hardware" schwierig.

Legt man als Lehrer den Unterrichtsschwerpunkt auf das Steuern der Modelle mit dem Computer, so tut man sicherlich gut daran, die Schüler auf das genaue Einhalten der Anleitungen zu verpflichten. Das genaue Arbeiten nach Vorlagen kann im Hinblick auf eine spätere Berufsausübung ein wichtiges Unterrichtsziel darstellen, welches in einigen Technik-Lehrplänen explizit gefordert wird.

Für den Einstieg in die Arbeit mit LEGO Lines empfiehlt sich dieses Vorgehen im Sinne des Prinzips der Isolation von Schwierigkeiten ebenfalls. Manche Schülergruppen bleiben sonst in der Bauphase stecken und kommen unter Umständen nie zur Programmierung ihres Modells. Zudem lassen sich die Strukturen von LEGO Lines unzweifelhaft leichter erarbeiten, wenn allen Schülergruppen identische Modelle zur Verfügung stehen und die gemachten Beobachtungen somit gleichermaßen für alle Gruppen zutreffen.

Für fortgeschrittene Schüler, die selbständiges Arbeiten gewohnt sind, ist demgegenüber der Bau und die Programmierung eines selbstentworfenen Modelles wesentlich reizvoller. Bei einem solchen freien Vorgehen

können die Vorlagen dann als Orientierungshilfe dienen und für die Lösung konstruktiver Details zu Rate gezogen werden.

Zum Antrieb der Modelle enthalten die Sets 4,5 Volt LEGO Motoren. Mit den ebenfalls vorhandenen Zahnrädern und Achsen können Getriebe gebaut und damit die Motorendrehzahl den Erfordernissen angepaßt werden.

Die LEGO Leuchtsteine können mit den farbigen Blenden zum Bau von Verkehrsampeln oder als Lichtschrankenelemente verwendet werden.

Die LEGO Optosensoren sind vielseitig einsetzbar. Sie reagieren auf kleinste Helligkeitsveränderungen und können so, wie bereits angedeutet, zusammen mit einem Leuchtstein als Lichtschranke verwendet werden. Da jeder Optosensor bereits eine Infrarotleuchtdiode eingebaut hat, kann er auch ohne externe Lichtquelle als Reflexsensor eingesetzt werden. Hierzu ist jedem Sensor eine Segmentscheibe mit abwechselnd hellen und dunklen Abschnitten beigefügt. Bringt man diese Segmentscheibe vor der ovalen Öffnung des Optosensors an, so werden beim Drehen der Scheibe die Hell-Dunkel-Wechsel registriert und der Computer kann damit die Umdrehungen einer Getriebeachse zählen.

Als Zubehör sind außerdem einfache Tastensensoren erhältlich, die als Endschalter Verwendung finden. Sie werden ebenfalls an die Eingänge 6 oder 7 des Interfaces angeschlossen und aktivieren bei gedrückter Taste den entsprechenden Eingabekanal. Im Gegensatz zu den Optosensoren brauchen diese mechanischen Sensoren zu einem sinnvollen Einsatz keine zusätzlichen Funktionselemente wie Leuchtsteine oder Segmentscheiben. Ihr Einsatz ist deshalb bei der erstmaligen Behandlung der Dateneingabe in den Rechner anzuraten; zudem können die Schüler die Funktion des Tastensensors, im wahrsten Sinne des Wortes, besser begreifen.

4 Arbeiten mit LEGO Lines

4.1 LEGO Lines starten

4.1.1 Start auf dem Commodore 64 und 128

Die ausgelieferte LEGO Lines Master Disk ist nicht kopiergeschützt. Schulen und andere Aus- und Weiterbildungseinrichtungen erwerben mit dem Kauf der Software nach Rücksendung einer beigelegten Antwortkarte das Recht, LEGO Lines für schulische Zwecke im eigenen Hause zu vervielfältigen. Sie sollten deshalb zuerst die Master Diskette in der benötigten Anzahl kopieren, die Originaldiskette dann an einem sicheren Ort aufbewahren und nur mit den Kopien arbeiten.

Das LEGO Kopierprogramm kann mit der Tastenfolge

```
LOAD "COPY LEGO LINES",8 <Return>
RUN <Return>
```

gestartet werden. Der Benutzer wird während des Kopiervorgangs durch Dialoganweisungen geführt.

Bei **ausgeschaltetem** Rechner verbinden Sie das Interface über das mitgelieferte Kabel mit dem USER-Port des Computers und versorgen das Interface über das separate Netzteil mit Strom. Weitere ausführliche Hinweise für die Installation und eine Anleitung für eine Fehlerdiagnose und -behebung erhalten Sie in der zum Interfacekabel mitgelieferten Gebrauchsanweisung.

Dann schalten Sie den Computer wie gewohnt ein und legen eine Kopie der Programmdiskette ins Laufwerk. Mit der Anweisung

```
LOAD "LEGO LINES",8 <Return>
```

wird das Ladeprogramm eingelesen und nach der READY-Meldung des Computers mit

```
RUN <Return>
```

gestartet. Für das nun folgende Laden und Starten des Programms müssen Sie etwas Geduld aufbringen. Es dauert aufgrund der niedrigen Datenübertragungsrate des Commodore-Laufwerks rund zwei Minuten,

bis sich LEGO Lines mit seiner graphischen Benutzeroberfläche meldet und Ihre Anweisungen entgegen nehmen kann.

Achtung: Falls Sie mit einem Commodore 128 arbeiten, muß dieser zuvor in den C64-Modus geschaltet werden. Dies geschieht durch Drücken der "Commodore"-Taste beim Einschalten des Computers.

4.1.2 Start auf einem MS-DOS-Computer

Die LEGO Lines Master Disk für IBM-Computer und Kompatible enthält mehrere Dateien oder Files, um mit den gängigen Bildschirmadaptern dieser Rechnerserie arbeiten zu können. Für eine Installation auf Ihrem eigenen System bzw. den Systemen Ihrer Schule ist deshalb nur ein Teil der vorhandenen Dateien notwendig.

Zunächst formatieren Sie eine leere Diskette als Systemdiskette und kopieren auch noch den deutschen Tastaturtreiber hinzu. Im mitgelieferten Lehrerhandbuch sind diese Schritte ausführlich beschrieben.

Das Zusammenstellen der richtigen Files können Sie sich von ebenfalls vorhandenen Stapel-Dateien abnehmen lassen. Arbeiten Sie mit einem Farbmonitor und einer CGA- oder EGA-Bildschirmkarte, so rufen Sie einfach

COCOPY.BAT <Return>

auf. Die LEGO Lines Programmdateien LINES.EXE, LINES2.EXE sowie die Dateien SCREEN.SCR und LINESDF.DAT werden damit auf Ihre Arbeitsdiskette kopiert. SCREEN.SCR enthält den Bildschirmaufbau der graphischen Oberfläche, in LINESDF.DAT sind die LEGO Lines Meldungen enthalten.

Verwenden Sie die genannten Bildschirmkarten jedoch mit einem Schwarzweißmonitor, so ist

SWCOPY.BAT <Return>

aufzurufen. Nun werden statt der vorgenannten Programmdateien die Dateien BWLINES.EXE und BWLINES2.EXE kopiert.

Enthält Ihr System eine Hercules-Karte, dann starten Sie einfach

HECOPY.BAT <Return>.

Jetzt wird zusätzlich noch das Emulationsprogramm EMU.COM kopiert, welches für eine korrekte Ansteuerung der Hercules-Karte sorgt.

Im ebenfalls angelegten Unterverzeichnis LINESFIL sind die LEGO Lines Beispielprogramme abgelegt. Auch die Steuerungsprogramme, die Sie selber schreiben werden, kommen beim Sichern automatisch in dieses Unterverzeichnis.

In allen drei Fällen wird zudem noch eine jeweils passende Version einer AUTOEXEC-Stapeldatei installiert, so daß die Arbeitsdiskette eigenstartfähig wird. Sie muß dann später nur noch vor dem Einschalten des Computers in das Bootlaufwerk eingelegt werden. Zunächst sollten Sie aber von Ihrer Schullizenz Gebrauch machen und die soeben erstellte Arbeitsdiskette in der benötigten Anzahl kopieren.

Arbeiten Sie mit einem Computer, der mit einer Festplatte ausgestattet ist, dann erzeugen Sie mit

MD \LEGO <Return>

ein passendes Unterverzeichnis auf Ihrer Festplatte und wechseln dann mit

CD \LEGO <Return>

in dieses Verzeichnis. Am einfachsten lassen Sie sich nun durch den DOS-Befehl

XCOPY A:*.* C:\LEGO /S <Return>

die gesamte Master Disk kopieren. Die obige Befehlssyntax setzt voraus, daß Sie auf Ihrer Festplatte einen Suchpfad auf das Unterverzeichnis mit den Betriebssystemdateien eingerichtet haben. Im Zweifel schlagen Sie bitte im DOS-Handbuch den richtigen Gebrauch des XCOPY-Befehles nach. Dieser Befehl hat den Vorteil, daß auch die Unterverzeichnisse LINESFIL und DRCK7BIT der Originaldiskette auf der Platte eingerichtet und die darin enthaltenen Files übertragen werden.

Wollen Sie Speicherplatz auf der Platte sparen, so können Sie die nicht benötigten Stapeldateien mit der Endung .BAT löschen. Welche Dateien sie letztendlich benötigen, hängt von der Ausstattung Ihres Computers ab:

Farbbildschirm mit CGA oder EGA-Bildschirmkarte:

LINES.EXE
LINES2.EXE
SCREEN.SCR
LINESDF.DAT

Monochrommonitor mit CGA oder EGA-Bildschirmkarte:

BWLINES.EXE
BWLINES2.EXE
SCREEN.SCR
LINESDF.DAT

Monochrommonitor mit Herculeskarte:

BWLINES.EXE
BWLINES2.EXE
SCREEN.SCR
LINESDF.DAT
EMU.COM

Sie können LEGO Lines direkt durch LINES.EXE bzw. BWLINES.EXE aufrufen. Bei der Arbeit mit einer Herculeskarte muß jedoch zuvor in jedem Fall das Emulationsprogramm EMU.COM gestartet werden! Mit diesen Angaben und den zusätzlichen Hinweisen im Handbuch sollte Ihnen die korrekte Konfiguration Ihres Systems nicht mehr schwer fallen.

Nach dem Einrichten der Software braucht nur noch die Hardware angeschlossen zu werden. Die Verbindung zwischen Rechner und Interface geschieht über eine mit dem Interfacekabel gelieferte Steckkarte. Diese wird bei **ausgeschaltetem** Computer in einen freien Steckplatz des Rechners gesteckt und das Interface über das Kabel mit der Buchse der Karte verbunden. Seine Stromversorgung erhält das Interface durch das externe Netzteil. Weitere ausführliche Hinweise für die Installation der Steckkarte und eine Anleitung für eine Fehlerdiagnose und -behebung erhalten Sie in der zum Interfacekabel mitgelieferten Gebrauchsanweisung.

Bei einem Diskettensystem legen Sie jetzt die zuvor erstellte Arbeitsdiskette ins Laufwerk A: und schalten den Computer dann ein. LEGO Lines wird nun vollautomatisch ohne weiteres Zutun geladen.

Bei der Arbeit mit einer Festplatte wechseln Sie nach dem Booten ins Verzeichnis \LEGO und rufen dort das für Ihr System zutreffende Programm auf. Auch diese Schritte können Sie mit etwas Systemkenntnis selbstverständlich durch eine entsprechende Stapeldatei Ihren speziellen Anforderungen anpassen und automatisieren.

4.2 Erste Schritte

Vorbemerkung: LEGO Lines verwendet auf Commodore-Computern und IBM-PC's sowie dazu kompatiblen Rechnern eine unterschiedliche Belegung der Funktionstasten. Bei der folgenden Beschreibung sind immer zuerst die Tastenbelegungen für IBM-Computer angegeben und die Tastenbelegung für den Commodore in Klammern angefügt!

Nach dem Laden und Starten präsentiert LEGO Lines seine graphische Benutzeroberfläche:

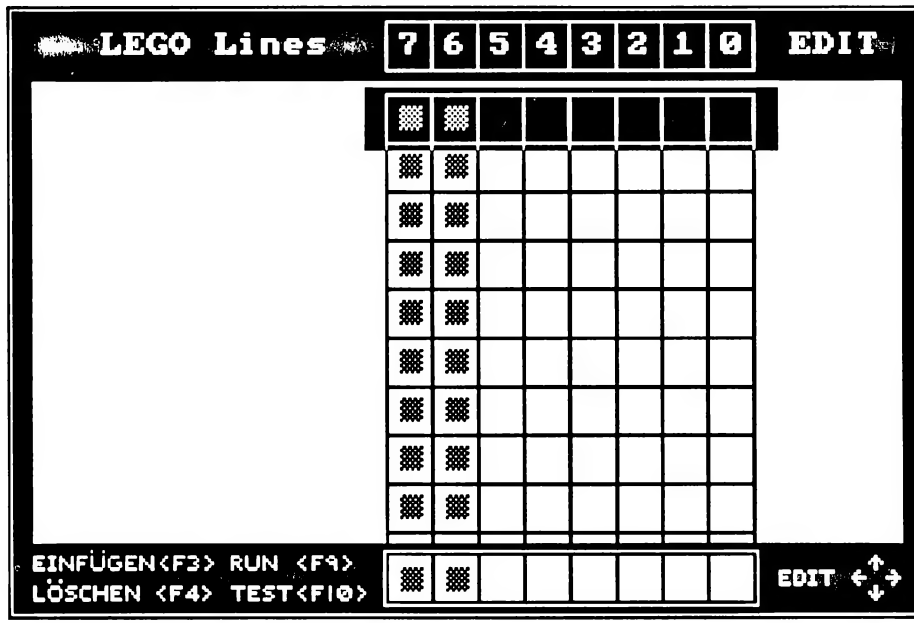


Abb. 4.1 LEGO Lines Benutzeroberfläche MS-DOS-Computer

Der Bildschirm ist in drei große Felder aufgeteilt. Links ist das Textfeld, in der Mitte, schachbrettartig gemustert, das Programmraster und rechts das Wertefeld. Das Eingabefeld, ein großer Cursorbalken, befindet sich unmittelbar nach dem Start in der ersten Zeile des Programmrasters. Es läßt sich einfach mit der Rechts- (<→>) und der Linkspfeiltaste (<←>) zwischen den einzelnen Spalten verschieben. Mit der Auf- (<↑>) und der

Abpfeiltaste (<↓>) kann es zudem in verschiedene Zeilen der jeweiligen Spalte gebracht werden. Am besten probieren Sie gleich die Funktion der Pfeiltasten kurz aus und setzen dann das Eingabefeld wieder in seine Ausgangsposition im Programmraster zurück.

Dieses Programmraster ist seinerseits in acht Spalten aufgeteilt. Sie sind von 0 bis 7 durchnumeriert, wobei sich die Spalten 6 und 7 durch ihre Schraffur abheben. Dadurch ergibt sich eine vollkommene Übereinstimmung mit dem Interface, auf dem die Buchsen 0 bis 5 als Ausgänge, die Buchsen 6 und 7 jedoch als Eingänge markiert sind.

Befindet sich das Eingabefeld im Programmraster, so werden nur Eingaben über die Tasten **<0>** bis **<5>** der Computertastatur entgegengenommen. Drücken wir beispielsweise auf die Taste **<5>**, so erhält die Spalte 5 des Programmrasters eine 1 eingetragen, während alle anderen Spalten 0 enthalten. Ein Druck auf die Taste **<3>** läßt nun auch in Spalte 3 eine 1 erscheinen. Ein erneuter Druck auf **<3>** schaltet wieder auf 0 zurück. Somit kann durch einfaches Drücken der Computertasten **<0>** bis **<5>** zwischen den beiden Zuständen 0 und 1 in der jeweiligen Spalte umgeschaltet werden. Dies erlaubt uns in einer Zeile des Programmrasters das Setzen eines gewünschten Bitmusters. Genauso lassen sich die nächsten Zeilen füllen, wenn vorher das Eingabefeld mit der <↓>-Taste weitergesetzt wurde. Die Spalten 0 bis 5 stellen damit die Ausgangskanäle unserer Steuerung beim Programmablauf dar; je nachdem, ob ein Kanal eine 1 oder eine 0 enthält, ist er ein- oder ausgeschaltet.

Beginnen wir nun mit einem ersten einfachen Programm. Zuvor sollten wir unseren Bildschirm wieder löschen und in den Ausgangszustand zurückversetzen. Dies kann ganz einfach mit **<Ctrl>-<F2>** (Commodore: **<CTRL>-<f3>**) geschehen. Da wir dadurch ein zuvor erstelltes Programm völlig löschen, fragt LEGO Lines nach, ob dies wirklich beabsichtigt ist. Wir beantworten diese Frage mit "ja", drücken **<Return>** und kommen wieder in den Ausgangszustand zurück.

In der ersten Zeile des Programmrasters schalten wir Kanal 0 durch Drücken der Taste **<0>** ein. Die nächste Zeile erreichen wir mit der <↓>-Taste und aktivieren dort Kanal 1. In der folgenden Zeile wird Kanal 2 eingeschaltet, in den nächsten Zeilen die Kanäle 3, 4 und 5.

Die einzelnen Programmzeilen können im Textfeld links vom Programmraster kommentiert werden. Dies ist zwar für einen Programmlauf nicht notwendig, Kommentare verbessern aber die Lesbarkeit von Programmen, so daß wir diese Möglichkeit gleich beim ersten Beispiel nutzen werden. Bewegen Sie hierzu zunächst das Eingabefenster nach links in die erste Zeile. Da die erste Zeile des Programms den Kanal 0 einschaltet, schreiben wir "Kanal 0 ein" als Kommentar in diese Zeile. Im Eingabefenster markiert ein kleiner Strich wie der gewohnte Cursor die aktuelle Schreibposition. Nach dem Eintrag unseres Kommentars können wir mit **<Return>** oder der **<↓>**-Taste in die nächsten Zeilen kommen, wo wir ebenfalls unsere Kommentare sinngemäß eintragen. Der Bildschirm sollte danach folgendes Programm enthalten:

	7	6	5	4	3	2	1	0	
Kanal 0 ein	*	*	0	0	0	0	0	1	
Kanal 1 ein	*	*	0	0	0	0	1	0	
Kanal 2 ein	*	*	0	0	0	1	0	0	
Kanal 3 ein	*	*	0	0	1	0	0	0	
Kanal 4 ein	*	*	0	1	0	0	0	0	
Kanal 5 ein	*	*	1	0	0	0	0	0	
	*	*							
	*	*							
	*	*							

Abb. 4.2 Programmraster LAUFLI1

Bitte beachten Sie, daß die tatsächlich auf dem Bildschirm angezeigte Anzahl von Programmrasterzeilen je nach verwendetem Computer und Bildschirmmodus von der obigen Darstellung abweichen kann. Wir werden deshalb in allen weiteren Darstellungen hier im Buch in jedem Falle genau die benötigten Zeilen angeben, völlig unabhängig davon, ob alle Zeilen auf dem Bildschirm angezeigt werden können oder nicht.

Dieses erste Programm ist gleichzeitig ein Beispiel für die einfachste Programmstruktur, die es gibt. Die lineare Abarbeitung aufeinanderfol-

gender Programmschritte nennt man eine Sequenz. Sie wird in einem Struktogramm durch eine Folge einfacher Rechtecke dargestellt:

Kanal 0 ein
Kanal 1 ein
Kanal 2 ein
Kanal 3 ein
Kanal 4 ein
Kanal 5 ein

Abb. 4.3 Struktogramm LAUFLI1

Zum Starten des Programmes dient die Funktionstaste **<F9>** (Commodore: **<f7>**). Ein Druck auf diese Taste läßt das Programm Zeile für Zeile abarbeiten, was auf dem Bildschirm und am Interface beobachtet werden kann. Auf dem Bildschirm fällt auf, daß das Eingabefenster in die erste Zeile des Textfeldes springt und dann Zeile für Zeile das gesamte Programm nach unten durchmarschiert. Nach der letzten Programmzeile springt das Eingabefenster wieder auf die erste Zeile zurück und bleibt dort stehen, der Programmlauf ist beendet.

Achtung: Ihr Programm wird nur gestartet, wenn sich das Eingabefenster irgendwo innerhalb des Programms und nicht auf einer noch leeren Programmzeile des Bildschirms befindet!

Dieses Mitlaufen des Eingabefensters stellt praktisch eine Trace-Funktion dar, da immer die gerade aktuelle Zeile markiert wird. Überzeugen Sie sich davon ruhig mit einem zweiten oder dritten Programmlauf. Vielleicht ging es Ihnen trotzdem noch zu schnell oder Sie fragen sich nun, was mit einem Steuerungsprogramm anzufangen ist, wenn die einzelnen Steuerschritte alle gleichschnell abgearbeitet werden.

Hier hilft uns das Wertefeld weiter. Dort kann für jede Programmzeile unter anderem deren Ausführungsdauer in Sekunden angegeben werden. Bewegen Sie das Eingabefenster ins Wertefeld der ersten Zeile und schreiben dort 1,0 hinein. Dies bedeutet, daß während des Programmlaufs diese Zeile genau eine Sekunde "ausgeführt" wird, also Kanal 0 diese Zeitspanne aktiviert bleibt, bevor die nächste Zeile an die Reihe

kommt. Diese Zeitintervalle können bis zu einer Genauigkeit von einer Zehntelsekunde in Dezimalschreibweise angegeben werden. LEGO Lines ist dabei so tolerant, daß es als Dezimal-Trennzeichen das Komma wie auch den Punkt akzeptiert. Die Werte 0.5 und 0,5 führen somit beide zu einer Ausführungsdauer von einer halben Sekunde. Es ist sinnvoll, Zeitangaben im Wertefeld als Dezimalbruch anzugeben, um sie von den dort ebenfalls möglichen Zählwerten zu unterscheiden. Versehen Sie nun alle Programmzeilen mit Zeitangaben von einigen Sekunden, die es Ihnen erlauben, die Programmausführung in Ruhe zu beobachten.

Beim erneuten Programmlauf läßt sich nun das Wandern des Eingabefensters besser verfolgen, und Sie stellen fest, daß der Marker in jeder Zeile genau die dort im Wertefeld angegebene Zeitspanne verbleibt. Achten Sie dann beim nächsten Programmlauf auf die allerletzte Programmrasterzeile am unteren Bildschirmrand innerhalb des dunklen Rahmens. Dies ist die Statuszeile. Hier wird die Bitkombination der gerade aktuellen Programmzeile eingeblendet. Beim Ablauf unseres Programms können wir sehen, wie die invers dargestellte "1", entsprechend den eingetragenen Zeiten, von Kanal 0 ganz rechts nach links wandert bis zum Feld für Kanal 5.

Ein Blick auf das angeschlossene Interface zeigt während des Programmlaufs genau das gleiche Phänomen: Nacheinander leuchten die Leuchtdioden von Ausgang 0 bis Ausgang 5 für die jeweils eingetragene Zeitspanne auf. Auch daran können wir wieder erkennen, daß nacheinander alle sechs Ausgabekanäle ein- und wieder ausgeschaltet werden, genau wie das im Programmraster festgelegt wurde.

Das Ergebnis unseres ersten Steuerungsprogramms erinnert stark an ein Lauflicht. Allerdings mit dem gravierenden Unterschied, daß ein Lauflicht in der Realität beliebig oft durchläuft, während unsere Version nach jedem Durchlauf neu gestartet werden muß. Es wäre schön, wenn wir unser Programm **immer wiederholen** lassen könnten. Tatsächlich gibt es in LEGO Lines eine entsprechende Strukturanweisung, die genau das Gewünschte macht, nämlich einen Programmblock aus aufeinanderfolgenden Zeilen **immer** zu **wiederholen**. Sie heißt auch genau so, nämlich WIEDERHOLE...IMMER. Die Anweisung WIEDERHOLE muß dabei genau vor der ersten Zeile und die Anweisung IMMER direkt nach der letzten Zeile des zu wiederholenden Programmblockes stehen. Die zu wiederholenden Zeilen werden somit von dieser Strukturanweisung

eingeklammert, was auch im zugehörigen Struktogramm zum Ausdruck kommt.

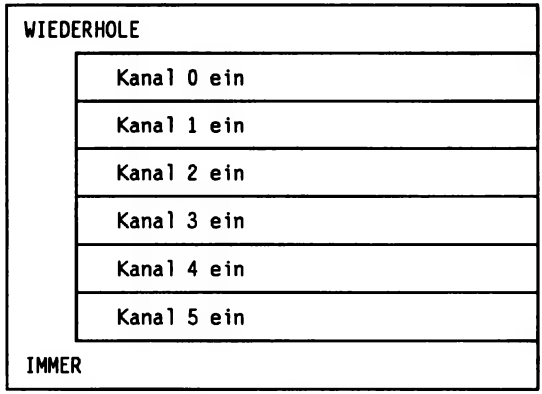


Abb. 4.4 Struktogramm LAUFLI3

Um diese Anweisung in unser Programm einbauen zu können, müssen wir vor der ersten Programmzeile Platz schaffen: Wir positionieren dazu das Eingabefenster in das Textfeld der ersten Zeile und drücken **<F3>** (Commodore: **<f1>**). Alle Programmzeilen werden um eine Position nach unten verschoben, die erste Zeile wird dadurch frei gemacht. Tippen Sie nun "wiederhole" (ohne Anführungsstriche) in Kleinbuchstaben ins Textfeld und achten darauf, was passiert, wenn Sie das letzte "e" eingeben: Die Anweisung wird in Großbuchstaben umgesetzt als Zeichen, daß LEGO Lines einen zulässigen "Befehl" erkannt hat. Dies dient nicht nur einer schnellen Überprüfung auf Tippfehler, sondern hilft auch, in der Textspalte Kommentare von STRUKTURANWEISUNGEN zu unterscheiden. Sie sollten deshalb möglichst nie Kommentare völlig in Großbuchstaben schreiben. Haben Sie sich mal vertippt, so können Sie mit der **<Rückpfeil>**-Taste (Commodore: **<INST-DEL>**) bereits eingegebene Zeichen wieder löschen. Eine komplette Programmzeile entfernen Sie einfach mit **<F4>** (Commodore: **<f3>**).

Durch den Eintrag einer Strukturanweisung in das Textfeld wird zudem das Programmraster in der zugehörigen Zeile gelöscht. Dies führt zu einer optischen Strukturierung des geschriebenen Programms und macht es dadurch leichter lesbar.

Wenn wir das so abgeänderte Programm jetzt starten wollen, so erhalten wir am unteren Bildschirmrand die Fehlermeldung:

WIEDERHOLE ohne BIS oder IMMER

Fehlermeldungen werden mit der **<Return>**-Taste quittiert. Dem Hinweis folgend, muß noch der zweite Teil der Strukturanweisung eingetragen werden. Also bewegen wir das Eingabefenster in die erste freie Zeile nach unserem Programm und schreiben dort das verlangte "immer". Auch hier werden, richtige Schreibweise vorausgesetzt, die Kleinbuchstaben wieder in Großbuchstaben umgewandelt. Nach dem Start läuft das Programm nun wie gewünscht in einer Endlosschleife. Abbrechen kann man jederzeit mit der Taste **<F1>** (Commodore: **<RUN-STOP>**).

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
Kanal 0 ein	*	*	0	0	0	0	0	1	1,0
Kanal 1 ein	*	*	0	0	0	0	1	0	0,5
Kanal 2 ein	*	*	0	0	0	1	0	0	1,0
Kanal 3 ein	*	*	0	0	1	0	0	0	0,5
Kanal 4 ein	*	*	0	1	0	0	0	0	1,0
Kanal 5 ein	*	*	1	0	0	0	0	0	0,5
IMMER									

Abb. 4.5 Programmraster LAUFLI3

Sie wissen sicherlich bereits, daß beim Ausschalten des Computers selbst erstellte Programme und Daten verlorengehen, wenn sie nicht zuvor auf externe Datenträger, beispielsweise auf Diskette oder Festplatte dauerhaft gespeichert worden sind. Das Speichern von Programmen ist bei LEGO Lines besonders einfach. Mit der Taste **<F5>** (Commodore: **<f2>**) wird auf eine andere Bildschirmdarstellung umgeschaltet, welche einen einfachen Diskettenzugriff ermöglicht.

Auf der linken Bildschirmhälfte werden die bereits vorhandenen LEGO Lines Programme angezeigt, auf der rechten Seite wird über ein kleines

Menü die gewünschte Aktion erfragt. Da wir unser Programm abspeichern, also sichern wollen, drücken wir auf die Zifferntaste **<2>**, gefolgt von **<Return>**. LEGO Lines erfragt dann den Namen, unter dem unser Programm abgespeichert werden soll und erwartet eine Bezeichnung, die aus maximal acht Zeichen bestehen darf. Die Namenseingabe wird ebenfalls mit **<Return>** abgeschlossen, unser Programm damit unter dem angegebenen Namen abgespeichert, und wir kommen automatisch wieder zur Programmieroberfläche zurück.

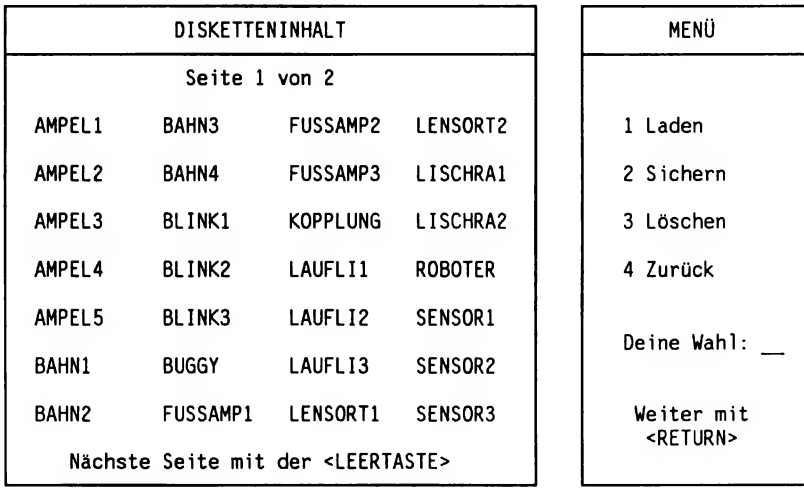


Abb. 4.6 Menü für Diskettenzugriff

Vielleicht möchten Sie Ihr erstes LEGO Lines Programm schwarz auf weiß besitzen? Auch dies ist bei angeschlossenem und eingeschaltetem Drucker kein Problem. Nach dem Druck auf die Funktionstaste **<F6>** (Commodore: **<Ctrl>-<f1>**) werden Sie gefragt, ob Ihr aktuelles Programm ausgedruckt werden soll. Bei einer positiven Antwort erhalten Sie Ihr Programmraster mit Strukturanweisungen, Kommentaren und den Angaben im Wertefeld als Listing auf Papier (Abb. 4.7). Besonders vorteilhaft ist hierbei, daß die Struktur des Programms in diesem Listing durch Einrücken der jeweiligen Ebenen deutlich hervorgehoben wird.

Falls Sie nun vorerst Ihre Arbeit mit LEGO Lines beenden möchten, so können Sie dies mit der Tastenkombination **<Ctrl>-<F8>** tun (Commodore: **<CTRL>-<f7>**). Diese Entscheidung müssen Sie aus Sicherheits-

gründen nochmals mit "ja" bestätigen, dann kommen Sie wieder auf die Betriebssystemebene zurück. Wenn Sie aber noch nicht aufhören möchten, antworten Sie mit "nein" und bearbeiten die folgende kleine Aufgabe:

Aufgabe LAUFLI4: Erweitern Sie das Lauflichtprogramm derart, daß das Aufleuchten der Leuchtdioden auf dem Interface nicht immer nur von rechts nach links, sondern auch wieder zurück nach rechts geschieht. Sie werden dabei feststellen, daß die maximal mögliche Länge von Steuerungsprogrammen nicht auf die Anzahl der am Bildschirm sichtbaren Programmzeilen beschränkt ist. Es sind bis zu 40 Programmzeilen möglich.

```
                    7 6 5 4 3 2 1 0
                    ---------------
WIEDERHOLE          . . . . . . . .
 Kanal 0 ein        * * 0 0 0 0 0 1   1,0
 Kanal 1 ein        * * 0 0 0 0 1 0   0,5
 Kanal 2 ein        * * 0 0 0 1 0 0   1,0
 Kanal 3 ein        * * 0 0 1 0 0 0   0,5
 Kanal 4 ein        * * 0 1 0 0 0 0   1,0
 Kanal 5 ein        * * 1 0 0 0 0 0   0,5
IMMER               . . . . . . . .
```

Abb. 4.7 LEGO Lines Programmausdruck

4.3 Projekt: Verkehrsampel

4.3.1 Vorüberlegungen zur Ampelsteuerung

Wenn es im Technikunterricht der Schule ans Steuern und Regeln geht, kommt meist die im täglichen Leben eines Verkehrsteilnehmers häufig gescholtene Verkehrsampel zu ungewohnten Ehren. Jedoch nicht ganz zu Unrecht, wie wir meinen: Es klang bereits an, daß sie aus dem heutigen Leben nicht mehr wegzudenken ist, sie gehört damit auch zum unmittelbaren Erfahrungsbereich der Schüler. Ihre Behandlung im Technikunterricht unterstützt grundsätzlich die fächerübergreifende Verkehrserziehung in der Schule. Besonders die Erörterung der einzelnen Ampelphasen und ihre notwendige Abstimmung zwischen den verschiedenen Ampeln einer Straßenkreuzung mag dem Schüler beim Programmieren einer Ampelsteuerung Erkenntnisse vermitteln, die ihm ein korrektes Verhalten im Straßenverkehr einsichtig werden lassen.

In technischer Hinsicht ist das Thema Ampelsteuerung für die Schule ebenfalls sehr gut geeignet, drei entsprechend gefärbte und angeordnete Lampen reichen für die Darstellung einer Verkehrsampel aus. Daß zu ihrer Steuerung nur drei Ausgabekanäle benötigt werden, reduziert die kognitiven Anforderungen bei der Programmieraufgabe und schafft damit Schülern den notwendigen Freiraum für die weitere Einarbeitung in die Benutzeroberfläche von LEGO Lines. Damit ist die Ampel ein geeignetes Einstiegsobjekt in die Thematik des Steuerns, wir wollen deshalb in diesem ersten Projekt einen möglichen Unterrichtsgang skizzieren.

Wer allerdings glaubt, aufgrund des genannten hohen Umweltbezugs im Unterricht sofort mit dem Steuern beginnen zu können, wird unter Umständen enttäuscht werden. Eigene Unterrichtserfahrungen haben gezeigt, daß die Abfolge der einzelnen Ampelphasen durchaus nicht allen Schülern geläufig ist. Wann GELB und wann ROT-GELB kommt, und ob es auch ein GRÜN-GELB gibt, darüber gehen die Schülermeinungen zum Teil bereits auseinander. Völlig uneinheitlich und verworren werden die Ansichten, wenn es um die Frage geht, welches denn die entsprechenden korrespondierenden Phasen der zweiten Kreuzungsampel sind. Hier lohnt sich sicherlich, nicht zuletzt auch im Hinblick auf den ebenfalls genannten Aspekt einer fächerübergreifenden Verkehrserziehung, ein Lerngang zur nächstgelegenen Verkehrsampel. Bei entsprechender Vorbereitung

können die dabei gewonnenen Erkenntnisse später direkt in das LEGO Lines Programmraster übertragen werden.

Beim Lerngang wird man natürlich die geplanten Beobachtungsaufgaben nach ihrem Schwierigkeitsgrad abstufen. Deshalb bietet sich zunächst die Beobachtung der Phasenfolge einer einzelnen Ampel an. Die gefundenen Ergebnisse werden an Ort und Stelle in ein vorbereitetes Raster eingetragen.

In diesem Raster ist für jede Ampelfarbe eine Spalte reserviert, in eine vierte Spalte soll die Zeitdauer der einzelnen Phasen geschrieben werden. Brennt beispielsweise das grüne Licht, so setzen die Schüler in das Kästchen unter der Bezeichnung "grün" ein Kreuz und vermerken in derselben Zeile in der letzten Spalte, wie lange die Grünphase gedauert hat. Die Gelb-Phase wird in die nächste Zeile durch ein Kreuz in die Spalte "gelb" eingetragen, die Dauer dieser Phase wird wieder in der letzten Spalte vermerkt. Ein Kreuz in der Spalte "rot" und ein entsprechender Zeiteintrag stellen in der nächsten Zeile den Rotabschnitt dar. Die nun folgende Rot-Gelb-Phase wird ebenfalls durch Kreuze in den entsprechenden Spalten festgehalten.

grün	gelb	rot	Zeit
X			30 s
	X		3 s
		X	15 s
	X	X	3 s
X			30 s
	X		3 s
		X	15 s
	X	X	3 s
X			30 s

Abb. 4.8 Beobachtungsraster Ampel

Recht schnell erkennen so die Schüler zum einen die korrekte Phasenabfolge und zum anderen die andauernde Wiederholung einer vierzeiligen Sequenz. Durch die Vorbereitung und Vorgabe des Beobachtungsrasters fällt die Übertragung der realen Ampelsteuerung auf die Modellebene mit LEGO Lines nicht mehr schwer.

Jedem Set LEGO Technic Control I sind sechs LEGO Leuchtsteine mit farbigen Blenden und weiterem Zubehör beigefügt, so daß im Unterricht recht schnell Modellampeln gebaut werden können. Man muß dann nur noch die Zuordnung der Leuchtsteine zu den Ausgängen des Interfaces festlegen und mit den ebenfalls mitgelieferten Kabeln die Verbindungen herstellen. Wir wollen hier verabreden, daß der grüne Leuchtstein an den Ausgang 5 angeschlossen wird. Gelb wird mit Ausgang 4 und Rot mit Ausgang 3 verbunden.

Nach dieser Festlegung und mit den notierten Beobachtungen fällt die Umsetzung in ein LEGO Lines Programm nicht mehr schwer. Für jedes Kreuz im Beobachtungsraster muß eine 1 im Programmraster eingetragen werden, dabei beschränken wir uns auf die Notation einer einzigen Ampelsequenz. Da die Spalten des Programmrasters mit den Kanalnummern fest vorbelegt sind, schreiben wir die Farben der jeweiligen Phase ins Textfeld.

Zur Kontrolle, ob die Zuordnung zwischen den Leuchtsteinen und den Interface-Ausgängen richtig erfolgt ist, können wir die Test-Funktion benützen. Durch einen Druck auf die Taste **<F10>** (Commodore **<f5>**) wird die Bitkombination der gerade markierten Programmrasterzeile in die Statuszeile am unteren Bildschirmrand übertragen und gleichzeitig werden die Ausgänge des Interfaces entsprechend geschaltet. Steht beispielsweise das Eingabefeld in der ersten Programmzeile, so muß nach dem Betätigen der Test-Taste das grüne Licht der Modellampel aufleuchten. Auf diese Weise geht man kurz alle Programmzeilen durch und überprüft so den korrekten Anschluß aller Leuchtsteine.

Zum Schluß werden die beim Lerngang gestoppten Zeiten ins Wertefeld geschrieben, wobei es notwendig sein kann, die in der Realität gefundenen Werte sinnvoll zu kürzen, um akzeptable Ausführungszeiten zu erreichen. Hat man das Programm bereits mit zu langen Ausführungszeiten gestartet, so kann es jederzeit mit der Taste **<F1>** (Commodore **<RUN-STOP>**) abgebrochen werden.

In der bisher vorliegenden Fassung wird die Ampelsequenz nach dem Start allerdings nur ein einziges Mal ablaufen, während sie in der Realität immer wiederholt wird. Die bereits bekannte Strukturanweisung WIEDERHOLE...IMMER löst dieses Problem, so daß unser erstes Ampelprogramm die folgende endgültige Form erhalten wird:

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
grün	*	*	1	0	0	0	0	0	10,0
gelb	*	*	0	1	0	0	0	0	1,0
rot	*	*	0	0	1	0	0	0	5,0
rotgelb	*	*	0	1	1	0	0	0	1,0
IMMER									

Abb. 4.9 Programmraster AMPEL1

Die WIEDERHOLE...IMMER-Anweisung läßt das Programm nach seinem Start in einer Endlosschleife ablaufen. Dies und die ausreichend lange Ausführungsdauer der einzelnen Anweisungen gibt Gelegenheit, die Protokollierung der einzelnen Schritte in den verschiedenen Ebenen exakt zu verfolgen und zu überdenken.

Im Programmraster auf dem Bildschirm wird wieder, genau wie schon vom Lauflicht-Programm gewohnt, die jeweils aktuelle Programmzeile durch das Eingabefeld markiert. Dieses Eingabefeld bewegt sich im Textfeld der Programmieroberfläche Zeile für Zeile weiter und hebt dadurch die als Kommentar eingetragene, momentan gültige Ampelphase deutlich hervor. Durch sinnvolles Ausnutzen der Kommentierungsmöglichkeiten erhält man so neben einer statischen Erklärung des Programms eine dynamische Überprüfungsmöglichkeit der einzelnen Anweisungen im Programmlauf. Ohne auf das Bitmuster im Programmraster achten und es "zurückübersetzen" zu müssen, ist man jederzeit über den momentanen Zustand der angeschlossenen Steuerung im Klartext informiert. Bei unerwarteten Ergebnissen werden beispielsweise ein markierter Kommentar "rot" und eine an der Modellampel aufleuchtende grüne Lampe sehr schnell zum Fehlerort und damit zu einer leichten Korrektur führen.

Das Bitmuster einer aktiven Programmzeile erscheint in der Statuszeile am unteren Bildschirmrand. Dies symbolisiert den Übergang zur Hardwareseite einer Computersteuerung. Der im Programmlauf hier sichtbare Wechsel zwischen 1 und 0 in den einzelnen Positionen entspricht ja genau dem Ein- und Ausschalten der jeweiligen Ausgabekanäle des Interfaces. Diese Übereinstimmung wird schließlich durch die graphische Gestaltung des Interfaces und durch die jedem Kanal zugeordnete Leuchtdiode verstärkt, das Bitmuster der Statuszeile findet sich an den roten Leuchtdioden der sechs Ausgabekanäle des Interfaces wieder.

Im Gegensatz zu Kalkulations- oder Dateiprogrammen enthalten Programme zur Prozeßsteuerung eine weitere Fehlermöglichkeit: Neben Fehlern im Programmcode, seien es logische Fehler oder einfach nur Tippfehler, kommt bei Steuerungsprogrammen die Hardwareseite als weitere Fehlerquelle hinzu. Eine falsche Verkabelung der angeschlossenen Modelle wird selbst bei einwandfreien Programmen unsinnige Ergebnisse liefern. LEGO Lines unterstützt mit seiner Transparenz beim Programmlauf die Einschränkung der Fehlerursache wesentlich und trägt so entscheidend mit dazu bei, daß Schüler ihre Programme selbst überprüfen und berichtigen können.

Nach diesen allgemeinen Überlegungen anhand unseres ersten einfachen Ampelprogramms ist es an der Zeit, in der Realität existierende Ampelsteuerungen auf die Modellebene abzubilden.

4.3.2 Die automatische Fußgängerampel

Den meisten Schülern dürften aufgrund ihres täglichen Schulwegs ampelgesteuerte Fußgängerüberwege bekannt sein. Das sicher vorhandene Wissen, daß die Fußgänger nur Grün erhalten können, wenn der Fahrzeugverkehr durch Rotlicht gestoppt ist, wird man ebenfalls bei dem im vorigen Abschnitt angesprochenen Lerngang präzisieren. Hierbei interessiert vor allem die genaue Abfolge der einzelnen Phasen beider Ampeln und ihre Abstimmung aufeinander. Die Ergebnisse werden wieder in einen vorbereiteten Beobachtungsbogen eingetragen. Für viele Schüler dürfte dabei die Erkenntnis neu sein, daß bei Fußgängerüberwegen die Grünphase der Fußgänger erst mit einer kleinen Verzögerung nach dem Rotlicht für die Autofahrer aktiviert wird. Außerdem erhält mit dem Ende der Grünphase für Fußgänger der Fahrverkehr nicht sofort

wieder freie Fahrt, um den noch auf dem Überweg befindlichen Fußgängern ein gefahrloses Überqueren zu ermöglichen. Den Schülern sollte bei der Diskussion dieser Punkte deutlich werden, daß diese Überlappungen der Rotphasen ihrer eigenen Sicherheit dienen und keinesfalls zu einer riskanten Straßenüberquerung mißbraucht werden dürfen!

Auto grün	Auto gelb	Auto rot	Fußg. grün	Fußg. rot	Zeit
X				X	30 s
	X			X	3 s
		X		X	1 s
		X	X		10 s
		X		X	5 s
	X	X		X	3 s

Abb. 4.10 Beobachtungsraster Automatische Fußgängerampel

Die Realisierung auf der Modellebene ist nach den bereits geleisteten Vorarbeiten einfach. Zur bereits vorhandenen Ampel für den Fahrverkehr wird mit zwei Leuchsteinen sowie einer roten und grünen Blende noch eine Fußgängerampel gebaut. Es wird zusätzlich vereinbart, daß die Ausgabekanäle 1 und 0 das Grün- und Rotlicht der Fußgängerampel versorgen sollen. Die Umsetzung des Beobachtungsrasters in die LEGO Lines Programmieroberfläche kann danach problemlos erfolgen. Das erstellte Programm AMPEL1 enthält bereits wesentliche Schritte der neuen Aufgabe. Um mit den Editiermöglichkeiten von LEGO Lines vertraut zu werden, wollen wir es als Basis für unsere weitere Arbeit benutzen. Ist Ihre Ampel noch in Betrieb, beenden Sie die Endlosschleife mit einem Druck auf die STOP-Taste **<F1>** (bzw. **<RUN-STOP>**).

Falls sie zwischendurch Ihre Arbeit mit LEGO Lines ganz unterbrochen hatten, so haben Sie sicherlich zuvor das Programm auf Diskette gesichert; in diesem Fall laden Sie es nun wieder in den Speicher: Mit der Taste **<F5>** (Commodore: **<f2>**) erhalten Sie Zugriff auf die Diskettenfunktionen von LEGO Lines. Die Funktion "Laden" erreichen Sie mit der Taste **<1>**, sie wird den Namen des zu ladenden Programmfiles erfragen. Geben Sie dann "AMPEL1" (ohne Anführungsstriche)

gefolgt von **<Return>** ein und LEGO Lines meldet sich wieder mit Ihrem ersten Ampelprogramm im Programmraster. Das Eingabefeld steht in der allerersten Zeile auf der WIEDERHOLE-Anweisung.

Mit der **<↓>**-Taste bringen wir das Eingabefeld in die nächste Zeile, wo es nun den Kommentar "grün" markiert. Die aktuelle Schreibposition wird durch einen kleinen Strich dargestellt. Mit der Leertaste rücken wir sie etwas nach rechts und schreiben die Ampelphase für die Fußgängerampel "ROT", zur Unterscheidung in Großbuchstaben, dazu. Mit der **<→>**-Taste bewegen wir dann das Eingabefeld ins Programmraster und aktivieren mit der Taste **<0>** den verabredeten Kanal für das Fußgänger-Rotlicht. Ob wir alles richtig gemacht haben, können wir sogleich mit der Test-Funktion (Taste **<F10>** bzw. **<f5>**) nachprüfen.

Mit den Pfeiltasten bewegen wir nun die Eingabemarkierung in das Textfeld der folgenden Zeile. Der Fahrverkehr erhält dort "gelb", die Fußgänger "ROT". Der Kommentar wird wieder ergänzt und im Programmraster der Kanal 0 aktiviert. In der nächsten Zeile zeigen beide Ampeln "rot"; auch hier werden im Textfeld und im Programmraster die notwendigen Ergänzungen vorgenommen.

Jetzt erst kann den Fußgängern grünes Licht gegeben werden. Dazu muß zunächst Platz für eine zusätzliche Zeile geschaffen werden. Setzen wir dazu den Marker auf die nächste Zeile mit dem Kommentar "rotgelb". An deren Stelle soll eine neue Zeile eingefügt werden, ein Druck auf die Taste **<F3>** (Commodore: **<f1>**) erledigt dies. Diese neue Zeile erhält den Kommentar "rot GRÜN", im Programmraster müssen die Kanäle 3 und 1 aktiviert werden. (Bitte beachten Sie, daß auf dem Commodore keine Umlaute vorhanden sind, sie müssen dort "GRUEN" eingeben!)

Wieder schließt sich eine Sicherheitsphase an, in der beide Ampeln Rotlicht zeigen. Wir fügen deshalb, wie eben gezeigt, nochmals eine neue Zeile ein, die den entsprechenden Kommentar erhält und in deren Programmraster die Kanäle 3 und 0 aktiviert werden. Für die nächste Phase können wir wieder auf Vorhandenes zurückgreifen. Der Fahrverkehr erhält in der letzten verbliebenen Zeile "rotgelb". Wir ergänzen "ROT" für die Fußgänger im Textfeld und im Programmraster. Am Schluß korrigieren und ergänzen wir noch die Zeitangaben im Wertefeld. Falsche Zahlenwerte ändern wir in Verbindung mit der **<Rückpfeil>**-

Taste (Commodore: **<INST-DEL>**-Taste). Unser Programm sollte nun folgende Gestalt haben:

		7	6	5	4	3	2	1	0	
WIEDERHOLE										
grün	ROT	*	*	1	0	0	0	0	1	10,0
gelb	ROT	*	*	0	1	0	0	0	1	1,0
rot	ROT	*	*	0	0	1	0	0	1	0,1
rot	GRÜN	*	*	0	0	1	0	1	0	3,0
rot	ROT	*	*	0	0	1	0	0	1	2,0
rotgelb	ROT	*	*	0	1	1	0	0	1	1,0
IMMER										

Abb. 4.11 Programmraster FUSSAMP1

Für einen späteren Rückgriff ist die Erstellung des zugehörigen Struktogramms sinnvoll:

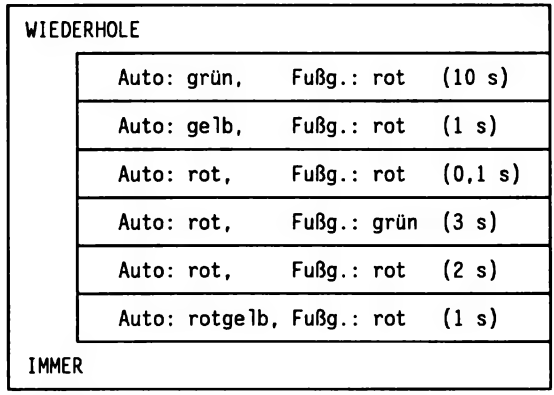


Abb. 4.12 Struktogramm FUSSAMP1

Haben wir bei der Änderung der Programmzeilen, bzw. beim Einfügen neuer Zeilen konsequent mit der Testfunktion die Richtigkeit unserer Eintragungen und Anschlüsse überprüft, so sollte der Programmstart eine automatische Steuerung eines Fußgängerüberweges zeigen. In festen

Zeitintervallen erhalten die Fußgänger Gelegenheit zum Überqueren der Straße. Diese zeitgesteuerte Schaltung findet sich meist bei größeren Straßenkreuzungen, an denen der Verkehr in einem festen Takt geregelt wird.

Von reinen Fußgängerüberwegen kennen die Schüler aber sicherlich eine bedarfsgeregelte Steuerung, wo der Fußgänger erst auf Anforderung ein Grünsignal zum Überqueren der Fahrbahn erhält. Zur Realisation dieser Steuerungsform auf Modellebene müssen wir unser seitheriges Konzept um einen wesentlichen Punkt erweitern. Bisher haben wir nur gesteuert, in unseren einfachen Beispielen Lampen ein- und ausgeschaltet. Der Informationstransport verlief dabei immer aus dem Computer heraus zum Interface, welches die empfangenen Signale auswertete und die angesprochenen Ausgabekanäle im gewünschten Sinne schaltete. Im Falle einer bedarfsgeregelten Fußgängerampel muß der Computer aber auch Signale von "außen" erhalten, um auf die Anforderung einer Grünphase reagieren zu können. Vor der Realisierung dieser Aufgabe sollten wir uns deshalb kurz mit den Voraussetzungen für die Signaleingabe in den Computer beschäftigen und die Möglichkeiten kennenlernen, die LEGO Lines zur Signalauswertung bietet.

4.3.3 Der Tastensensor als Eingabeelement

Alle Signale unserer Umwelt lassen sich mit Hilfe der Physik beschreiben. Der Klang einer Autohupe gelangt als Schalldruck an unser Ohr, das Rotlicht einer Ampel wird als elektromagnetische Welle von unserem Auge aufgenommen. Zur Signalaufnahme ist es daher notwendig, physikalische Größen zu erfassen und in eine Form zu bringen, die eine Weiterverarbeitung zuläßt. Bei den Lebewesen haben spezialisierte Sinnesorgane diese Ausgabe übernommen: Unsere Augen erfassen einen bestimmten Ausschnitt elektromagnetischer Wellen, unsere Ohren verarbeiten Schall eines bestimmten Frequenzintervalls. Alle unsere Sinnesorgane wandeln die aufgenommenen unterschiedlichen physikalischen Größen in Nervenimpulse um, die unser Gehirn verarbeiten kann. Zur Signalverarbeitung im Computer müssen die verschiedenenen physikalischen Größen unserer Umwelt ebenfalls erfaßt und in ein elektrisches Signal umgewandelt werden. Diese Aufgabe übernehmen spezielle Meßwertaufnehmer, sogenannte Sensoren, die es heute für nahezu alle beobachtbaren physikalischen Größen gibt. Während bei

industriellen Einsätzen häufig eine möglichst genaue quantitative Meßwerterfassung nötig ist, beispielsweise die genaue Erfassung einer Temperatur, reicht für unsere Zwecke eine rein qualitative Signalaufnahme aus. Bei der Programmierung eines bedarfsgesteuerten Fußgängerüberweges genügt einzig und allein die Feststellung, ob auf einen Knopf gedrückt wurde oder nicht.

Zu den LEGO Technic Control Sets sind Tastensensoren als Zubehör erhältlich, die für die gestellte Aufgabe hervorragend geeignet sind. Schließen wir deshalb beide Sensoren mit je einem Kabel an je eine Eingangsbuchse eines betriebsbereiten Interfaces an. Ein Druck auf die graue Taste eines Sensors bringt die dem Eingabekanal zugeordnete Leuchtdiode auf dem Interface zum Aufleuchten und zeigt damit die Aktivierung dieses Eingabekanals an. Auf dem Bildschirm wird jedoch noch keine Änderung sichtbar, da wir uns im sogenannten Editiermodus befinden. Zum Umschalten in den Direktmodus dient die Taste **<F7>** (Commodore: **<f6>**). Im Direktmodus wird die Statuszeile aktiviert, in dem beim Aufruf in alle Positionen eine 0 eingetragen wird. Jedes Ansprechen eines Ausgabekanals über die Computertastatur erfolgt nun sofort. Ein Druck auf die Taste **<5>** setzt eine 1 an die entsprechende Bitposition der Statuszeile, wie wir das bereits vom Editiermodus her gewohnt sind. Jetzt wird aber die Anweisung, Kanal 5 einzuschalten, sofort ausgeführt, wie an der brennenden Leuchtdiode auf dem Interface und am Grünlicht der angeschlossenen Ampel ersichtlich ist. Ein erneuter Druck auf die Zifferntaste schaltet den Kanal wieder aus. Der Direktmodus stellt damit eine weitere Testfunktion der Ausgabekanäle zur Verfügung. Aber auch der Zustand der Eingabekanäle wird nun in den Positionen 6 und 7 der Statuszeile angezeigt. Beim Betätigen der Tastensensoren wird eine 1 in die zugehörige Position geschrieben, beim Loslassen erscheint wieder die 0. Erneutes Drücken der Taste **<F7>** (Commodore: **<f6>**) schaltet in den Editiermodus zurück.

4.3.4 Strukturanweisungen zur Eingabeabfrage

Zur Auswertung der Eingabekanäle enthält LEGO Lines verschiedene Strukturanweisungen. Sie ermöglichen durch die Abfrage der Eingabesignale im Gegensatz zur unbedingten und fortwährenden Ausführung der WIEDERHOLE...IMMER-Schleife eine bedingte Bearbeitung festgelegter Programmschritte.

Es ist zunächst sicher wünschenswert, die Wiederholung einer Programmsequenz vom Vorliegen bestimmter Bedingungen abhängig zu machen. Eine Programmsequenz soll beispielsweise **wiederholt** werden, **bis** eine zuvor festgelegte Bedingung erfüllt ist. LEGO Lines stellt hierfür die Strukturanweisung WIEDERHOLE...BIS zur Verfügung. Die mit "BIS" formulierte Bedingung kann natürlich nur im Zusammenhang mit den Eingabekanälen 6 und 7 formuliert werden. Am besten schauen wir uns dies gleich an einem kleinen Beispiel an: Unsere Ampel soll gelb blinken, **bis** der Eingabekanal 6 durch einen Druck auf den angeschlossenen Sensor aktiviert wird. Im Struktogramm kann das Programm folgendermaßen dargestellt werden:

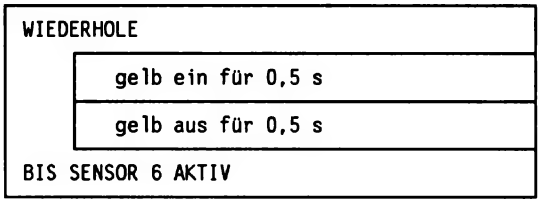


Abb. 4.13 Struktogramm BLINK1

Die zu wiederholenden Anweisungen werden eingeklammert. Jedoch wird die eingeklammerte Sequenz nicht mehr "IMMER", also endlos wiederholt, sondern nur, bis der Sensor 6 aktiv ist. Diese Abbruchbedingung wird so auch im Struktogramm formuliert.

Zum Erstellen dieses Programmes löschen wir zuerst den Programmspeicher mit der Tastenkombination **<Ctrl>-<F2>** (Commodore: **<CTRL>-<f3>**). In die erste Zeile schreiben wir die WIEDERHOLE--Anweisung, in der zweiten schalten wir durch Aktivieren des Ausgabekanals 4 das gelbe Licht für eine halbe Sekunde ein und in der dritten Zeile für dieselbe Zeitspanne wieder aus. In die nächste Zeile schreiben wir "bis" (ohne Anführungszeichen) in Kleinbuchstaben. Wie gewohnt wird nach dem Tippen des letzten Buchstabens das Wort in Großbuchstaben umgesetzt und im Programmraster verschwinden alle Ausgabepositionen, so daß nur noch auf die Spalten 6 und 7 des Programmrasters zugegriffen werden kann. Da der Pegel des Eingabekanals 7 nicht interessiert, blenden wir diesen Kanal durch zweimaliges Drücken der Taste **<7>** aus. Die entsprechende Position des Programmrasters wird dadurch schraffiert dargestellt. Unsere Ampel soll blinken, bis der Eingabekanal 6 über einen Sensor **aktiviert** wird, deshalb tragen wir durch einen Druck auf die Taste

<6> eine 1 an dieser Position ein. Damit haben wir die Abbruchbedingung angegeben: Die WIEDERHOLE-Schleife wird solange ausgeführt, BIS der am Eingabekanal 6 angeschlossene Sensor aktiviert wird.

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
gelb ein	*	*	0	1	0	0	0	0	0,5
gelb aus	*	*	0	0	0	0	0	0	0,5
BIS	*	1							

Abb. 4.14 Programmraster BLINK1

Da nach der Zeile mit der BIS-Anweisung keine weiteren Anweisungen mehr formuliert sind, kann dieses Blinklichtprogramm mit dem Tastensensor beendet werden. Mit der Strukturanweisung WIEDERHOLE...BIS steht somit eine Abbruchbedingung zur Verfügung, um kontrolliert einzelne Programmteile zu verlassen oder, wie hier, das ganze Programm zu beenden.

Vielleicht ist Ihnen beim Programmlauf aufgefallen, daß das Blinken nicht wie beabsichtigt genau symmetrisch erfolgt, sondern die Dunkelphase merklich länger dauert. Dies wird von der Trace-Funktion des mitlaufenden Markers verursacht, da Bildschirmausgaben ebenfalls Zeit benötigen. Zu den angegebenen 0,5 Sekunden Dunkelphase kommt noch diejenige Zeitspanne dazu, die benötigt wird, um den Marker auf die BIS-Zeile und von dort an den Anfang auf die WIEDERHOLE-Anweisung zu setzen. Um dennoch ein symmetrisches Blinkverhältnis zu erhalten, muß die angegebene Zeitdauer der Dunkelphase entsprechend verkürzt werden. Dies ist beim Commodore die einzige Korrekturmöglichkeit, da er nicht über die Ablauffunktion der IBM-PC-Version verfügt. Auf IBM-PC's und Kompatiblen haben wir mit **<F9>** seither immer den sogenannten Sichtlauf-Modus aktiviert, in welchem der Programmablauf durch das Mitwandern des Markers auf dem Bildschirm sichtbar ist. Die Vorteile des Sichtlaufs zur Fehlersuche wurden bereits angedeutet. Fehlerfreie Programme und besonders solche für zeitkritische Aufgaben startet man besser durch **<Shift>-<F9>** im Ablaufmodus, um ein realistisches Zeitverhalten zu erreichen.

Bedingungen werden im täglichen Sprachgebrauch häufig auch durch ein **"wenn"** charakterisiert. Man sagt beispielsweise: "Wenn die Sonne scheint, gehen wir spazieren". Die Aktion "spazieren gehen" wird damit vom Vorliegen einer bestimmten Bedingung, dem Scheinen der Sonne abhängig gemacht. Diese Formulierung hat fast unverändert Eingang in die Programmiersprachen gefunden, man denke nur an die IF...THEN-Anweisung von BASIC oder an die IF...ELSE-Anweisung in C. Die Strukturanweisung von LEGO Lines heißt entsprechend WENN...ENDE WENN. Die Bedingung, deren Vorliegen durch das "WENN" von LEGO Lines überprüft wird, muß wieder in den Spalten 6 und 7 des Programmrasters angegeben werden. Sehen wir uns diese Strukturanweisung ebenfalls an einem Beispiel an. Diesmal gehen wir den umgekehrten Weg, indem wir ein vorgegebenes Programm analysieren:

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
grün	*	*	0	0	0	0	1	0	0,5
WENN	*	1							
rot	*	*	0	0	0	0	0	1	0,5
ENDE WENN									
IMMER									

Abb. 4.15 Programmraster BLINK2

Eine WIEDERHOLE...IMMER-Schleife klammert alle Anweisungen ein. Die Anweisung "grün" wird dabei bei jedem Schleifendurchlauf 0,5 Sekunden lang ausgeführt. Wenn der Eingabekanal 6 aktiviert ist, wird anschließend auch das rote Licht der Fußgängerampel für eine halbe Sekunde eingeschaltet, bevor der Ablauf von neuem beginnt. Im zugehörigen Struktogramm (Abb. 4.16) wird der Programmablauf graphisch weiter verdeutlicht.

Nachdem das grüne Licht eingeschaltet wurde, wird geprüft, ob der Sensor 6 aktiv ist. Da diese Abfrage genau zwei mögliche Ergebnisse haben kann, werden im Struktogramm beide Alternativen graphisch dargestellt. Bei positivem Abfrageergebnis wird die "ja"-Spalte durchlaufen und das Rotlicht für 0,5 Sekunden eingeschaltet. Bei negativem Ergebnis

erfolgt keine Handlung, da die "nein"- Spalte keinen Eintrag enthält, der Programmlauf beginnt wieder von vorne. Wenn der Sensor nicht aktiviert ist, erfolgt keine Aktion, deshalb wird nur die grüne Lampe andauernd brennen. Wenn er jedoch aktiviert ist, kommt es zu einem Blinken von Rot und Grün an der Fußgängerampel. Die WENN...ENDE WENN-Anweisung bietet somit die Möglichkeit, Alternativen im Programmlauf zu realisieren.

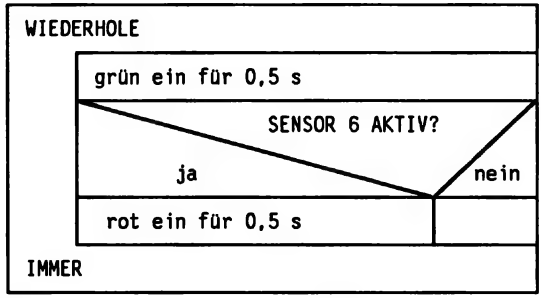


Abb. 4.16 Struktogramm BLINK2

Aufgabe BLINK3: Wandeln Sie das Programm BLINK2 durch die Veränderung einer Strukturanweisung so ab, daß der Programmlauf durch Aktivierung des Sensors am Eingabekanal 7 abgebrochen wird!

Beide Strukturanweisungen zur Realisation von bedingten Verzweigungen wurden kurz hintereinander und damit quasi gleichzeitig vorgestellt, die einfache Syntax und Verwendung beider Anweisungen rechtfertigt dieses Vorgehen hier im Buch. Für die Einführung im Unterricht ist ein anderer Weg jedoch sinnvoller. Ziel muß es sein, daß die Schüler beide Strukturanweisungen im Hinblick auf ihren Gebrauch und ihre Einsatzmöglichkeiten klar unterscheiden können. Eine Diskrimination setzt aber unabdingbar das Erfassen und Beschreiben der zu unterscheidenden Dinge voraus. Es ist daher ratsam, im Unterricht erst eine Strukturanweisung vorzustellen und ihre Verwendung an mehreren Beispielen aufzuzeigen, bevor die zweite Anweisung behandelt wird. Die Verwendung von Beispielprogrammen erspart dabei wortreiche Erklärungen und ermöglicht den Schülern durch die Beobachtung des Markers im Sichtlaufmodus ein analytisches Begreifen der Strukturanweisungen. Durch eine geschickte Wortwahl bei der Aufgabenstellung und -besprechung lassen sich zudem weitere Hilfen einbauen.

Beispiel SENSOR1: Mit den Eingabekanälen 6 und 7 soll zwischen dem roten und dem grünen Leuchtstein der Fußgängerampel umgeschaltet werden: **Wenn** Sensor 6 aktiviert wurde, soll der rote Leuchtstein brennen, **wenn** Sensor 7 aktiv ist, soll auf den grünen Leuchtstein umgeschaltet werden.

Bereits durch die Wortwahl in der Aufgabenstellung wird die Verwendung der Strukturanweisung WENN...ENDE WENN nahegelegt.

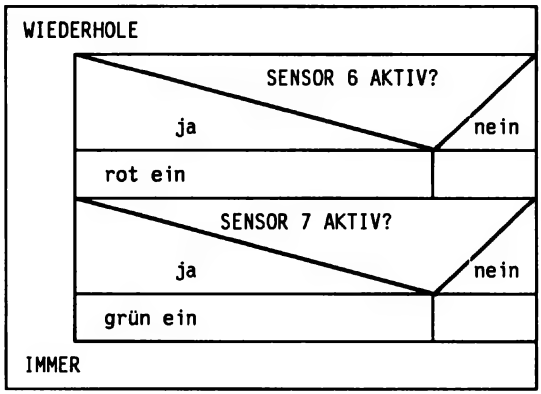


Abb. 4.17 Struktogramm SENSOR1

Innerhalb der äußeren WIEDERHOLE...IMMER-Schleife werden fortlaufend beide Sensoren kurz nacheinander abgefragt. Bei aktiviertem Eingabekanal 6 wird entsprechend der formulierten Bedingung das rote Licht der Fußgängerampel eingeschaltet. Selbst wenn man den Tastensensor an Kanal 6 daraufhin wieder losläßt, brennt die rote Lampe weiter, da LEGO Lines den zuletzt aktivierten Zustand der Ausgabekanäle solange beibehält, bis es wieder auf eine nächste Steueranweisung stößt. Solange aber kein Sensor betätigt wird, d.h. keine der beiden WENN-Bedingungen erfüllt ist, überspringt der Marker die Zeilen "rot" und "grün". Erst beim Betätigen des Sensors an Kanal 7 wird die Ausgabeanweisung in der Zeile "grün" erreicht und die Fußgängerampel auf Grün umschalten.

Natürlich kann ein aktivierter Eingabekanal nur erkannt werden, wenn sich der Marker zu diesem Zeitpunkt auf der entsprechenden WENN-Abfrage befindet. Eine kurze Sensorbetätigung wird deshalb nicht registriert, wenn sich der Programmzeiger in diesem Moment beispielsweise auf der

WIEDERHOLE-Anweisung befindet. Deshalb muß im Sichtlaufmodus der Sensor solange betätigt werden, bis die Ampel umschaltet. Im Ablaufmodus der PC-Version dagegen erfolgt praktisch eine unmittelbare Reaktion.

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
WENN	*	1							
rot	*	*	0	0	0	0	0	1	
ENDE WENN									
WENN	1	*							
grün	*	*	0	0	0	0	1	0	
ENDE WENN									
IMMER									

Abb. 4.18 Programmraster SENSOR1

Das obige Programm simuliert eine einfache Speicherschaltung. Ein kurzer Druck auf einen Sensor wird dauerhaft festgehalten. Auch lange Zeit später noch ist durch die eindeutige Zuordnung der Farben zu den Eingabekanälen feststellbar, welcher Sensor zuletzt aktiviert wurde. Diese Information bleibt solange bestehen, bis sie durch Betätigen des anderen Sensors überschrieben wird.

Beispiel SENSOR2: Jedes Betätigen von Sensor 6 soll zu einer Farbumschaltung an der Fußgängerampel führen.

Diese knappe Formulierung läßt die Aufgabe zunächst schwierig erscheinen. Im Unterrichtsgespräch wird man deshalb die Aufgabenstellung nochmals diskutieren und dabei vielleicht zu der folgenden oder einer ähnlichen Beschreibung kommen: Das grüne Licht soll leuchten, **bis** der Sensor betätigt wird. Danach soll das rote Licht leuchten, **bis** ebenfalls wieder der Sensor aktiviert wird, worauf dann wieder auf Grün umgeschaltet wird.

Mit dieser Formulierung hat man dann bereits den zugrundeliegenden Algorithmus in seiner Grobstruktur erfaßt. Das fortlaufende Abfragen legt die Verwendung einer äußeren WIEDERHOLE...IMMER-Schleife nahe, die überlegte Wortwahl der erarbeiteten Aufgabenformulierung impliziert den Gebrauch von WIEDERHOLE...BIS-Anweisungen. Die gefundene Grobstruktur des Algorithmus kann dann im Struktogramm verfeinert werden, dessen Umsetzung in ein Programm nicht mehr schwer fällt.

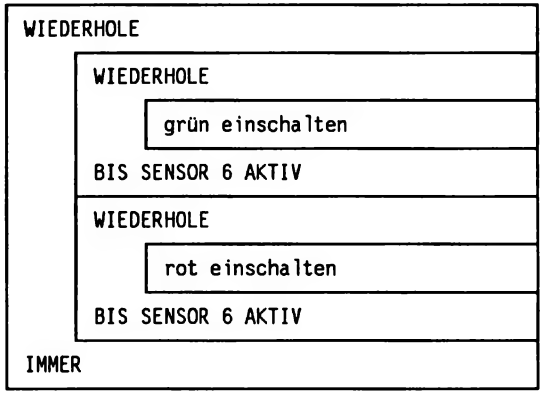


Abb. 4.19 Struktogramm SENSOR2

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
WIEDERHOLE									
grün ein	*	*	0	0	0	0	1	0	
BIS	*	1							
WIEDERHOLE									
rot ein	*	*	0	0	0	0	0	1	
BIS	*	1							
IMMER									

Abb. 4.20 Programmraster SENSOR2

Mit diesem Programm haben wir übrigens eine bistabile Kippstufe realisiert. Eine bistabile Kippstufe ist eine Schaltung, die zwei Zustände aufweisen kann, in unserem Fall entweder "Rot" oder "Grün". Jedes "Triggern" läßt die Schaltung in den anderen Zustand kippen, in welchem sie dann stabil bis zum nächsten Triggerimpuls verharrt.

In der Digitalelektronik und damit natürlich auch in Computern spielen solche Kippschaltungen, die im Englischen lautmalerisch Flip-Flop genannt werden, eine große Rolle. In den unterschiedlichsten Ausführungen benötigt man sie zumeist für Speicherzwecke. Zwei Formen haben wir mit den beiden obigen Beispielen kennengelernt, eine dritte Variante wird in der nächsten Aufgabe vorgestellt.

Beispiel SENSOR3: Programmieren Sie eine monostabile Kippstufe. Die monostabile Kippstufe kennt ebenfalls zwei unterschiedliche Zustände, in unserem Fall wieder "Grün" und "Rot". Sie besitzt aber nur einen stabilen Zustand ("Grün"), in dem sie andauernd verharren kann. Ein Triggern kippt die Schaltung in den instabilen Zustand ("Rot"), in welchem sie eine bestimmte Zeitspanne (2 Sekunden) verbleibt, bis sie von selbst wieder in den stabilen Zustand zurückkippt.

4.3.5 Die Fußgängerampel mit Anforderung

Im letzten Abschnitt "Strukturanweisungen zur Eingabeabfrage" haben wir uns nun das notwendige Wissen zur Bewältigung unserer ursprünglich gestellten Aufgabe angeeignet. Für die Programmierung des bedarfsgesteuerten Fußgängerüberweges haben wir sogar zwei Strukturanweisungen kennengelernt, mit deren Hilfe die Aufgabe gelöst werden kann. Daß wir somit auf zwei Wegen zum Ziel kommen können, wird bereits in den verschiedenen Formulierungen deutlich, mit denen das Problem genauer umrissen wird. Die Beschreibung: "Am Fußgängerüberweg soll der Fahrverkehr Grün erhalten, **bis** ein Fußgänger auf den Anforderungsknopf drückt" impliziert bereits durch die getroffene Wortwahl den Gebrauch der WIEDERHOLE...BIS-Anweisung. Die Verwendung der Anweisung WENN...ENDE WENN wird durch eine andere Formulierung nahegelegt, beispielsweise: "**Wenn** der Knopf gedrückt wird, soll die Ampel umschalten".

Die Vielfalt der möglichen Lösungswege ist beim Programmieren am Computer die Regel, und diese Erkenntnis sollte den Schülern anhand der gestellten Aufgabe vermittelt werden. Wahrscheinlich dürften bei der Besprechung der Aufgabe im Unterrichtsgespräch ohnehin beide Möglichkeiten genannt werden. Da man in der Diskussion die Vor- und Nachteile der jeweiligen Lösungen jedoch kaum klar herausstellen kann, bietet sich ein arbeitsteiliges Vorgehen an, in welchem beide Lösungen von unterschiedlichen Gruppen programmiert werden. Um jedem Schüler jedoch auch die andere Vorgehensweise vor Augen zu führen, sollten beide Struktogramme im Klassenplenum erarbeitet werden.

Als Grundlage dafür kann das bereits erarbeitete Struktogramm der automatischen Ampelsteuerung (Abb. 4.12) dienen, das bereits fast alle notwendigen Steuerungsanweisungen enthält und in welches lediglich noch die Strukturanweisungen zur Sensorabfrage eingebaut werden müssen. Da in Wirklichkeit die Phasenumschaltung meist nicht sofort mit dem Druck des Fußgängers auf den Anforderungsknopf, sondern etwas verzögert erfolgt, wird man für eine realistische Lösung noch die entsprechende Steuerungsanweisung zusätzlich einbauen. Für die beiden Lösungen ergeben sich dann die folgenden Struktogramme FUSSAMP2 (Abb. 4.21) und FUSSAMP3 (Abb. 4.22).

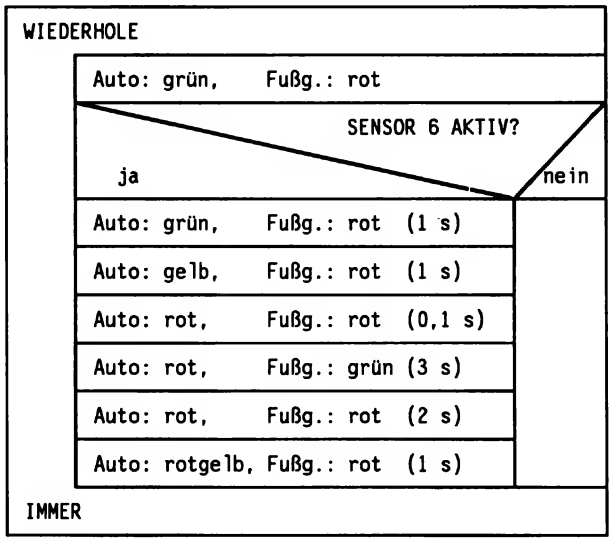


Abb. 4.21 Struktogramm FUSSAMP2

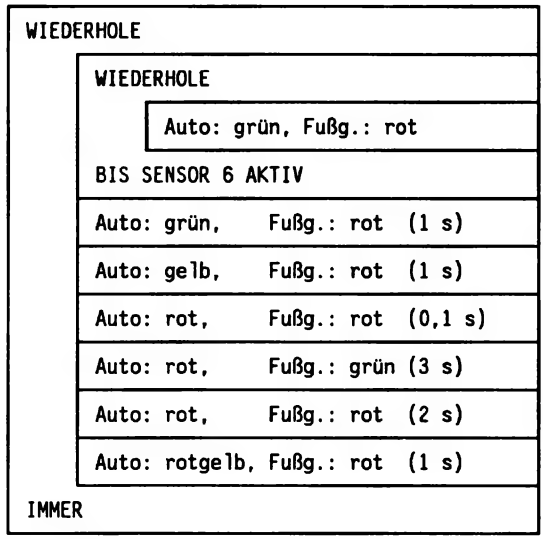


Abb. 4.22 Struktogramm FUSSAMP3

Nach dem gemeinsamen Erarbeiten dieser Struktogramme versuchen die verschiedenen Schülerarbeitsgruppen jeweils eine Lösung in ein LEGO Lines Programmraster umzusetzen. Um Schreibarbeit zu sparen, ist es sinnvoll, das abgespeicherte Programm FUSSAMP1 der automatischen Ampelsteuerung von der Diskette wieder in den Arbeitsspeicher zu laden und es durch Einfügen einiger Zeilen mit der Taste **<F3>** (Commodore: **<f1>**) an die neuen Anforderungen anzupassen.

Dabei darf auch nicht vergessen werden, die im ursprünglichen Programm festgelegte Zeitdauer von 10 Sekunden der ersten "grün ROT"-Phase zu entfernen, da sonst die Auswertung des Eingabekanals nur in entsprechenden Zeitintervallen stattfinden kann.

Die Umsetzung der Struktogramme in den verschiedenen Arbeitsgruppen sollte die in den Rastern FUSSAMP2 (Abb. 4.23) und FUSSAMP3 (Abb. 4.24) gezeigten Ergebnisse liefern.

Bei beiden Programmen erfolgt ein Umschalten der Ampelphasen nur nach dem Aktivieren des Sensors. Wenn damit auch beide Versionen die gestellten Anforderungen erfüllen, so gibt es doch eine bessere und eine schlechtere Lösung. Beim Austesten beider Programme werden die

Schüler sicherlich bald feststellen, daß bei der Lösung mit der Anweisung WENN...ENDE WENN eine kurze Sensorbetätigung recht häufig verloren gehen kann, bzw. zum Umschalten der Ampelphasen der Sensor ziemlich lange gedrückt bleiben muß. Das Programm FUSSAMP3 reagiert dagegen recht schnell auf eine Aktivierung des Sensors.

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
grün ROT	*	*	1	0	0	0	0	1	
WENN	*	1							
grün ROT	*	*	1	0	0	0	0	1	1,0
gelb ROT	*	*	0	1	0	0	0	1	1,0
rot ROT	*	*	0	0	1	0	0	1	0,1
rot GRÜN	*	*	0	0	1	0	1	0	3,0
rot ROT	*	*	0	0	1	0	0	1	2,0
rotgelb ROT	*	*	0	1	1	0	0	1	1,0
ENDE WENN									
IMMER									

Abb. 4.23 Programmraster FUSSAMP2

Verfolgt man bei der Programmausführung den Weg des Markers, so läßt sich die Ursache unschwer erkennen: Bei der Strukturanweisung WENN...ENDE WENN muß der Marker bei nicht aktiviertem Eingangssignal bis ans Programmende und von dort ganz an den Anfang zurückspringen, bis er schließlich die Abfrageanweisung wieder erreicht. Dadurch, daß das Programm in seiner vollen Länge nicht auf dem Bildschirm dargestellt werden kann, kommt zu der Zeit, die für die Bewegung des Programmzeigers erforderlich ist, auch noch die Zeit für das sogenannte Scrollen des Bildschirms dazu. Während dieser Zeit kann natürlich kein Eingabekanal ausgewertet werden, der Sensor muß entsprechend lange gedrückt bleiben.

Ganz anders jedoch bei der Version FUSSAMP3: Hier springt der Marker nur zwischen den nahe beieinanderliegenden Anweisungen WIEDER-

HOLE und BIS hin und her, so daß eine Sensorabfrage in wesentlich kürzeren Zeitintervallen erfolgen kann. Im Sichtlauf der IBM-PC-Version und bei der Arbeit am Commodore dürfte dies daher die bessere Vorgehensweise sein.

Mit einer im vorigen Abschnitt beim Programm SENSOR1 angesprochenen Arbeitsweise von LEGO Lines läßt sich sogar noch eine weitere kleine Geschwindigkeitssteigerung herausholen: Wie wir dort erfuhren, behält LEGO Lines den Schaltzustand der Ausgabeleitungen bei, bis es auf eine neue Ausgabeanweisung stößt.

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
WIEDERHOLE									
grün ROT	*	*	1	0	0	0	0	1	
BIS	*	1							
grün ROT	*	*	1	0	0	0	0	1	1,0
gelb ROT	*	*	0	1	0	0	0	1	1,0
rot ROT	*	*	0	0	1	0	0	1	0,1
rot GRÜN	*	*	0	0	1	0	1	0	3,0
rot ROT	*	*	0	0	1	0	0	1	2,0
rotgelb ROT	*	*	0	1	1	0	0	1	1,0
IMMER									

Abb. 4.24 Programmraster FUSSAMP3

Es besteht daher die Möglichkeit, die Anweisungszeile "grün ROT" aus der WIEDERHOLE...BIS-Schleife herauszunehmen und ihr voranzustellen. Die entsprechende Ampelphase bleibt dann solange bestehen, bis der Eingabekanal 6 aktiviert wird. Beim Warten auf dieses Eingabesignal muß der Marker jetzt nur noch zwischen zwei Zeilen hin- und herspringen, was entsprechend schnell erledigt werden kann.

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
gruen ROT	*	*	1	0	0	0	0	1	
WIEDERHOLE									
BIS	*	1							
gruen ROT	*	*	1	0	0	0	0	1	1,0
gelb ROT	*	*	0	1	0	0	0	1	1,0
rot ROT	*	*	0	0	1	0	0	1	0,1
rot GRUEN	*	*	0	0	1	0	1	0	3,0
rot ROT	*	*	0	0	1	0	0	1	2,0
rotgelb ROT	*	*	0	1	1	0	0	1	1,0
IMMER									

Abb. 4.25 Programmraster FUSSAMP3, Commodore-Version

4.3.6 Steuerung einer Kreuzungsampel

Der Programmierung einer Ampelanlage an einer Straßenkreuzung sollte zunächst wieder die genaue Beobachtung einer real existierenden Anlage vorausgehen. Erschwert wird dieses Vorhaben dadurch, daß es meist schwierig ist, von einem Standpunkt aus beide korrespondierenden Ampeln beobachten zu können. Hier hat sich die Aufteilung in Gruppen und eine Verständigung über Handzeichen oder Zuruf bewährt. Trotzdem wird es einiger Phasenwechsel bedürfen, bis schließlich das vorbereitete Beobachtungsraster korrekt ausgefüllt ist.

Wieder fällt auf, daß, genau wie bei der Fußgängerampel, die Phasen nicht genau parallel verlaufen. Mit dem Rotlicht für eine Straßeneinmündung erhält der Querverkehr nicht gleichzeitig Grün, sondern muß sich im Interesse einer höheren Verkehrssicherheit noch mit einer kurzen Rotgelbphase gedulden.

Ampel 1			Ampel 2			Zeit
grün	gelb	rot	GRÜN	GELB	ROT	
X					X	30 s
	X			X	X	3 s
		X		X	X	0,5 s
		X	X			10 s
	X	X		x		3 s
	X	X			X	0,5 s

Abb. 4.26 Beobachtungsraster Automatische Kreuzungsampel

Die im Modell vorhandene Fußgängerampel wird durch einen weiteren Leuchtstein mit gelber Blende zur Ampel 2 umgebaut. Das Anschlußkabel ihres grünen Leuchtsteins wird in den Ausgang 2 des Interfaces umgesteckt, damit Ausgang 1 für den Anschluß des neu hinzugekommenen gelben Leuchtsteins frei wird.

```
WIEDERHOLE
    grün     ROT      (5 s)
    gelb     ROTGELB  (1 s)
    rot      ROTGELB  (0,2 s)
    rot      GRÜN     (3 s)
    rotgelb  GELB     (1 s)
    rotgelb  ROT      (0,2 s)
IMMER
```

Abb. 4.27 Struktogramm AMPEL2

Der Übertrag der gemachten Beobachtungen in das LEGO Lines Programmraster dürfte keine höheren Anforderungen an die Schüler mehr stellen und hat somit eher Übungscharakter. Zur Vereinheitlichung sollte zuvor verabredet werden, daß die Kommentare für Ampel 1 im

Textfeld links in Kleinbuchstaben und diejenigen für die zweite Ampel rechts in Großbuchstaben eingetragen werden. Sinnvoll ist natürlich auch eine entsprechende Verkürzung der Zeitintervalle. Nach dem Testen und eventuell notwendigem Korrigieren wird das Programm unter der Bezeichnung AMPEL2 auf Diskette abgespeichert. Die damit erarbeitete und in Abb. 4.27 nochmals in ihrer Struktur dargestellte Programmierung einer Ampelanlage an einer Straßenkreuzung soll im folgenden in zwei Versionen erweitert werden.

4.3.7 Umschaltung auf Nachtbetrieb

Bei nur schwach frequentierten Kreuzungen wird die Ampelanlage in den späten Abendstunden auf Nachtbetrieb umgeschaltet. Die Ampeln für die vorfahrtberechtigte Straße werden dabei in der Regel abgeschaltet, der wartepflichtige Querverkehr erhält blinkend Gelb. In den frühen Morgenstunden wird dann wieder auf den bekannten Tagesbetrieb zurückgeschaltet. Dieses Umschalten geschieht meistens derart, daß beide Straßen zunächst für einige Sekunden Dauergelb erhalten und anschließend auf Rot wechseln. Der nachrangige Verkehr bleibt auf Rot, während auf der vorfahrtberechtigten Seite gleich darauf über Rotgelb auf Grün umgeschaltet wird. Aus dieser Konstellation kann dann problemlos der Tagesbetrieb wieder aufgenommen werden. Da Lerngänge zu diesen Zeiten wohl schwerlich durchführbar sein dürften, muß diese Information vom Lehrer geliefert werden. Die einzelnen Blöcke sind bereits vorhanden: AMPEL2 liefert den Tagesbetrieb; das gelbe Blinklicht der Nachtphase wurde im Programm SENSOR1 erstellt. Diese beiden Programmteile müssen somit nur noch durch entsprechende Strukturanweisungen verbunden und durch die Umschaltphasen ergänzt werden.

In der Realität werden beide Betriebsarten zeitgesteuert umgeschaltet. Für ein Nachempfinden auf der Modellebene ist es sicherlich ausreichend, die Dauer einer Betriebsart durch die Angabe der Anzahl von Sequenzwiederholungen anzugeben. Dies erfordert neben der Möglichkeit zum fortwährenden Wiederholen in einer Endlosschleife (WIEDERHOLE...IMMER) und einem Wiederholen, bis eine bestimmte Bedingung erfüllt ist (WIEDERHOLE...BIS), noch eine weitere Strukturanweisung, die eine festlegbare Anzahl von Wiederholungen ermöglicht. Die Programmiersprache BASIC hält für diese Schleifenbildung die Anweisung FOR...NEXT bereit, in LEGO Lines wurde dafür die Strukturan-

weisung WIEDERHOLE n...ENDE WIEDER implementiert. Für "n" muß im Wertefeld die Anzahl der gewünschten Wiederholungen stehen. Am besten schauen wir uns den Gebrauch dieser neuen Strukturanweisung gleich in einem Beispiel an:

	7	6	5	4	3	2	1	0	
WIEDERHOLE									10
GELB ein	*	*	0	0	0	0	1	0	0,5
GELB aus	*	*	0	0	0	0	0	0	0,5
ENDE WIEDER									

Abb. 4.28 Programmraster BLINK1 (verändert)

Hier ist das Programm BLINK1 verändert, indem der ehemalige Strukturanweisungsteil BIS durch ENDE WIEDER ersetzt wurde. Die Angabe, wie oft die durch WIEDERHOLE und ENDE WIEDER eingeklammerte Sequenz zu wiederholen ist, schreibt man in das Wertfeld der WIEDERHOLE-Zeile. Unser Beispiel führt zu einem zehnmaligen Blinken des gelben Leuchtsteins. Bei der Darstellung der Strukturanweisung WIEDERHOLE n...ENDE WIEDER im Struktogramm weichen wir von der im LEGO Unterrichtsbegleitmaterial vorgeschlagenen Notierung ab, da die hier propagierte Schreibweise eher der Darstellung im Programmraster entspricht.

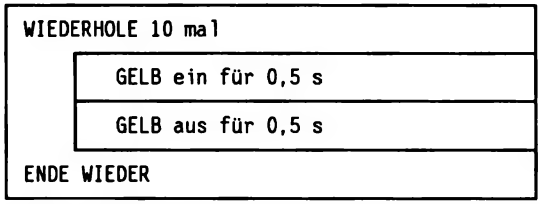


Abb. 4.29 Struktogramm BLINK1 (verändert)

Die Lösung der ursprünglichen Aufgabe kann nun ebenfalls zunächst im Struktogramm vorbereitet und dargestellt werden. Als Ausgangspunkt dient das bereits vorhandene Struktogramm AMPEL2, dessen äußere WIEDERHOLE...IMMER-Schleife durch die Anweisung WIEDERHOLE 10...ENDE WIEDER ersetzt wird. Daran schließt sich sofort das veränderte Struktogramm BLINK1 an. Ergänzt werden müssen lediglich

einige Steuerungsanweisungen zum Umschalten in den Tagesbetrieb. Für einen fortlaufenden Betrieb wird das Ganze schließlich noch in eine Endlosschleife gepackt.

```
WIEDERHOLE
    WIEDERHOLE 10 mal
        grün     ROT      (5 s)
        gelb     ROTGELB  (1 s)
        rot      ROTGELB  (0,2 s)
        rot      GRÜN     (3 s)
        rotgelb  GELB     (1 s)
        rotgelb  ROT      (0,5 s)
    ENDE WIEDER
    WIEDERHOLE 10 mal
        GELB ein für 0,5 s
        GELB aus für 0,5 s
    ENDE WIEDER
    gelb     GELB (2 s)
    rot      ROT  (1 s)
    rotgelb  ROT  (1 s)
IMMER
```

Abb. 4.30 Struktogramm AMPEL3

Dieses vom Umfang her bereits recht beachtliche Programm enthält dennoch nicht viel Neues; zwei bereits erarbeitete Module können fast unverändert übernommen werden. Hier wird deutlich, daß die Bezeichnung "Strukturanweisung" für LEGO Lines Befehlswörter durchaus zu recht verwendet wird, da sie in der Tat eine Strukturierung des Programms ermöglichen. Die Aufteilung eines Programms in einzelne Module ist einer der Hauptgedanken der strukturierten Programmierung Neben der Übersichtlichkeit und Transparenz der so erstellten Program-

me treten an diesem Beispiel durch den Rückgriff auf bereits vorhandene Module vor allem die Annehmlichkeiten bei der Softwareerstellung zutage.

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
WIEDERHOLE									10
grün ROT	*	*	1	0	0	0	0	1	5,0
gelb ROTGELB	*	*	0	1	0	0	1	1	1,0
rot ROTGELB	*	*	0	0	1	0	1	1	0,2
rot GRÜN	*	*	0	0	1	1	0	0	3,0
rotgelb GELB	*	*	0	1	1	0	1	0	1,0
rotgelb ROT	*	*	0	1	1	0	0	1	0,2
ENDE WIEDER									
WIEDERHOLE									10
GELB ein	*	*	0	0	0	0	1	0	0,5
GELB aus	*	*	0	0	0	0	0	0	0,5
ENDE WIEDER									
GELB gelb	*	*	0	1	0	0	1	0	2,0
ROT rot	*	*	0	0	1	0	0	1	1,0
ROTGELB rot	*	*	0	1	1	0	0	1	1,0
IMMER									

Abb. 4.31 Programmraster AMPEL3

LEGO Lines bietet leider keine Möglichkeit, bereits vorhandene Module von einer Diskette oder der Festplatte einzubinden. Wir behelfen uns, indem wir, ausgehend vom umfangreichsten Programmteil, die zusätzlichen Anweisungen ergänzen. Laden Sie dazu das Steuerungsprogramm AMPEL2 in den Speicher. Verändern und ergänzen Sie es dann entsprechend dem obigen Programmraster AMPEL3.

Ins Wertefeld der ersten WIEDERHOLE-Anweisung tragen Sie die Zahl 10 ein und ersetzten die Anweisung IMMER durch die nun notwendige Form ENDE WIEDER. Um Strukturanweisungen zu ändern, muß die betreffende Zeile zunächst gelöscht und dann gegebenenfalls durch Einfügen einer neuen Zeile der Platz für die korrekte Anweisung geschaffen werden. Löschen Sie deshalb die Zeile mit der Strukturanweisung IMMER am Programmende und ersetzen Sie sie durch die Anweisung ENDE WIEDER. Die veränderte Form des Programms BLINK1 kann dann sofort angefügt werden, ebenso die drei Anweisungszeilen zum Zurückschalten in den Tagesbetrieb. Unsere Ampelsteuerung soll in einer Endlosschleife laufen. Schließen Sie deshalb die angefügten Zeilen mit der Anweisung IMMER ab und schaffen am Programmanfang mit einer neuen Zeile Platz für die nötige WIEDERHOLE-Anweisung.

Wohl gibt das jetzige Programm eine zwar modellhaft verkürzte, aber doch bereits recht realistische Abfolge der Ampelsteuerung im Tagesverlauf wieder. Trotzdem wird vielen Schülern aufgefallen sein, daß viele Kreuzungen nicht nach einer starren Zeitscheibe, sondern verkehrsabhängig gesteuert werden. Als letzte Aufgabe in diesem Projekt soll deshalb eine bedarfsabhängige Ampelsteuerung realisiert werden.

4.3.8 Einsatz des Optosensors

Die Steuerung solch flexibler Ampelanlagen erfolgt kontaktlos über im Boden verlegte Induktionsschleifen. Häufig kann man die Lage dieser Schleifen in den einzelnen Fahrspuren an den für ihre Einbringung notwendigen Sägespuren im Asphalt noch ausmachen. Durch den hohen Metallgehalt eines darüber befindlichen Autos wird die Induktivität der als Spule wirkenden Schleife verändert. Diese Veränderung wiederum wird von der Ampelsteuerung als Anforderung zur Phasenumschaltung registriert.

Kontaktlose Steuerungen üben auf viele Menschen einen eigenartigen Reiz aus, auch Schüler sind meist begierig zu erfahren, wie denn solch eine Induktionsschleife funktioniert. Als Lehrer sollte man deshalb diese Motivation seiner Schüler aufgreifen, zumal das Funktionsprinzip recht einfach mit in jedem Physiksaal vorhandenen Material demonstriert werden kann.

Eine Spule mit 1200 Windungen wird an den Wechselstromausgang eines Experimentiertrafos angeschlossen und mit 15 Volt beschickt. In diesen Stromkreis kommt zusätzlich noch ein Amperemeter als Anzeigegerät. Nach dem Einschalten des Trafos wird das Amperemeter bald eine konstante Stromstärke von etwa 0,8 Ampere anzeigen. Bringt man jedoch einen eisernen Gegenstand nahe über die Spulenöffnung, so zeigt das Meßgerät eine veränderte, etwas niedrigere Stromstärke an. Diese Änderung kann von einer geeigneten Schaltung als Anforderung zur Phasenumschaltung interpretiert werden.

Das LEGO Technic Control Set bietet zwar keine Induktionsschleife als Eingabesensor an, jedoch kann für eine kontaktlose Steuerung der vorhandene Optosensor verwendet werden. Der im Sensor enthaltene Fototransistor wird beim Anschluß an das betriebsbereite Interface auf den Umgebungslichtpegel abgestimmt und reagiert dann auf Helligkeitsschwankungen. Diese Lichtveränderungen werden, genau wie das Betätigen des Schalters auf dem Tastensensor, an das Interface zurückgemeldet. Die hochempfindliche Elektronik des Sensors sorgt dabei für eine sehr feinfühlige Ansprache.

Für das Verständnis der Wirkungsweise des Optosensors schließen Sie ihn am besten gleich an eine Eingangsbuchse des Interfaces an und richten seine kleine ovale Öffnung gegen ein helles Fenster oder eine Lichtquelle im Zimmer. Wenn Sie dann mit den aufgefächerten Fingern einer Hand in wenigen Zentimetern Entfernung vor der Sensoröffnung entlangfahren, werden die dadurch hervorgerufenen Helligkeitsunterschiede durch das Flackern der grünen Leuchtdiode am Interfaceeingang angezeigt. Wenn zudem LEGO Lines mit der Taste **<F7>** (Commodore: **<f5>**) in den Direktmodus geschaltet ist, dann wird die Aktivierung des beschalteten Eingabekanals in der Statuszeile am unteren Bildschirmrand ebenfalls sichtbar.

Bei diesem ersten Experimentieren mit dem Optosensor werden Sie sicher auch dessen bereits erwähnte sensible Reaktion erfahren. Bereits kleinste Lageveränderungen reichen unter Umständen für die Generierung des jeweils anderen Signalpegels aus. Beim Betrieb in Steuerungsanlagen sind solche Zufälligkeiten natürlich unerwünscht. Man muß deshalb darauf achten, daß der Sensor nicht durch Streulicht irritiert wird und ihn möglichst nur bei genau definierten Lichtpegeln einsetzen. Dies kann beispielsweise sehr einfach durch den Aufbau einer Lichtschranke

geschehen. Plaziert man den Sensor und einen Leuchtstein so, daß das Licht des Lämpchens aus einigen Zentimetern Entfernung auf die ovale Sensoröffnung trifft, so wird jede Unterbrechung des Lichtstrahls durch das Aktivieren des Eingabekanals angezeigt.

Zur Auswertung der von einem Optosensor generierten Signale können natürlich die bereits bekannten Strukturanweisungen WENN...ENDE WENN und WIEDERHOLE...BIS verwendet werden. Führen wir uns die Programmierung einer Lichtschranke gleich an einem einfachen Beispiel vor Augen: Ein grüner Leuchtstein soll brennen, solange der Lichtstrahl unserer Lichtschranke nicht unterbrochen wird. Eine Unterbrechung des Strahls und ihre Dauer soll durch einen roten Leuchtstein angezeigt werden. Dieses Funktionsprinzip von Lichtschranken kennen Schüler von Aufzugtüren. Es kann auch in Alarmanlagen eingesetzt werden.

Für die Abfrage von Eingangspegeln stehen uns bisher die oben genannten Strukturanweisungen zur Verfügung. Beide können auch hier wieder zur Lösung der Aufgabe herangezogen werden. Da die korrekte Realisierung bedingter Verzweigungen im Programmlauf manchen Schülern durchaus Schwierigkeiten bereiten, bietet sich gerade an diesem einfachen Beispiel nochmals eine Wiederholung und Aufarbeitung der gegebenen Möglichkeiten an. Dabei wird es sinnvoll sein, die beiden möglichen Lösungen jeweils anhand ihrer Struktogramme im Klassenverband gemeinsam zu erarbeiten.

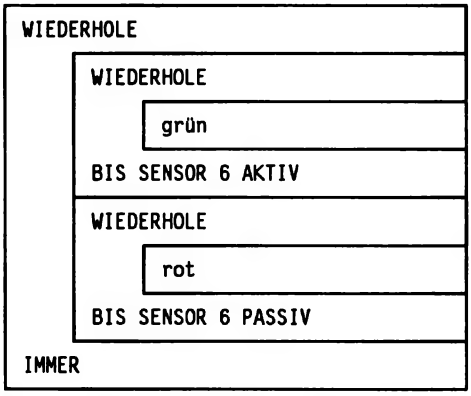


Abb. 4.32 Struktogramm LISCHRA1

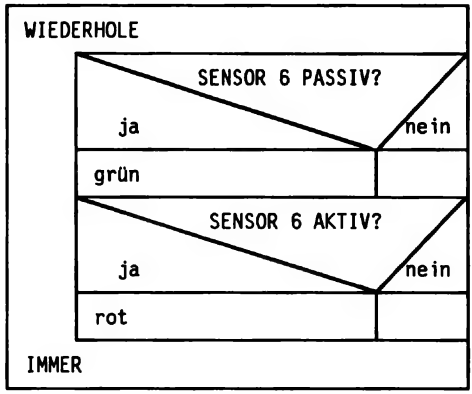


Abb. 4.33 Struktogramm LISCHRA2

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
grün	*	*	0	0	0	1	0	0	
WIEDERHOLE									
BIS	*	1							
rot	*	*	0	0	0	0	0	1	
WIEDERHOLE									
BIS	*	0							
IMMER									

Abb. 4.34 Programmraster LISCHRA1

Danach können beide Lösungen im arbeitsteiligen Verfahren von verschiedenen Schülergruppen in entsprechende LEGO Lines Programme umgesetzt werden. Neu ist dabei, daß nun erstmals ein Eingabekanal auf einen nicht aktivierten Zustand hin abgefragt wird. Dazu muß an der Position für Kanal 6 der entsprechenden Strukturanweisung eine 0 eingetragen sein. Wenn wir davon ausgehen, daß die Modellampeln noch am Interface angeschlossen sind, können die Ausgangskanäle 2 und 0 zum Ansteuern des roten und grünen Leuchtsteins verwendet werden.

Spätestens beim direkten Vergleich beider Programme werden die Vorteile des ersten Lösungsansatzes deutlich. Wie bereits beim Programm für die Steuerung der Fußgängerampel FUSSAMP3 (Abb. 4.24) ist wieder die Verwendung der Strukturanweisung WIEDERHOLE...BIS vorteilhafter, da hier der Markierungsbalken beim Sichlauf auf dem Bildschirm kleinere Strecken zurücklegen muß. Dieser kleinere Weg bedeutet aber letztlich eine höhere Ausführungs- und Reaktionsgeschwindigkeit des Programms.

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
WENN	*	0							
grün	*	*	0	0	0	1	0	0	
ENDE WENN									
WENN	*	1							
rot	*	*	0	0	0	0	0	1	
ENDE WENN									
IMMER									

Abb. 4.35 Programmraster LISCHRA2

4.3.9 Bedarfsgeregelte Kreuzungssteuerung

Beenden wollen wir unser Ampel-Projekt mit einem schon recht komplexen Programm für eine vollautomatische und bedarfsgeregelte Ampelsteuerung einer einfachen Straßenkreuzung. Für die Aufgabenstellung legen wir eine Kreuzung zwischen einer Hauptverkehrs- und einer Nebenstraße zugrunde. Der Verkehr auf der Nebenstraße soll dabei nur auf Anforderung eine kurze Grünphase erhalten; andernfalls soll der Hauptverkehr ungehindert fließen können. Dies entspricht fast vollständig der Aufgabenstellung bei der Fußgängerampel. Den dort gefundenen Lösungsweg kann man mit Ausnahme einer kleinen Änderung in der Abfolge der Phasenumschaltung vollständig in das neue Programm übernehmen.

Bis auf die Verwendung eines Optosensors in einer Lichtschranke anstatt des Tastensensors zur Entgegennahme der Umschaltanforderung böte diese Aufgabenstellung allerdings noch nicht viel Neues. Deshalb vereinbaren wir zusätzlich eine vollautomatische Umschaltung der Ampelsteuerung zwischen Tages- und Nachtbetrieb in Abhängigkeit vom Tageslicht. Das Programm für den Nachtbetrieb ist ebenfalls schon vorhanden, es besteht genau genommen aus zwei Teilen: der erste ist für das Blinken verantwortlich, der zweite bewirkt die Umschaltung zurück vom Nacht- auf den Tagesbetrieb.

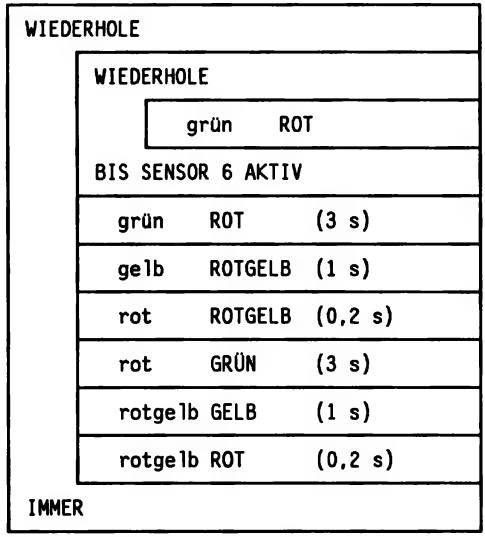


Abb. 4.36 Programmteil Tagesbetrieb

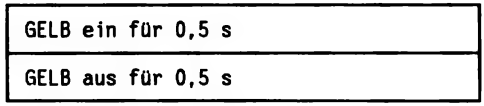


Abb. 4.37 Programmteil Blinken

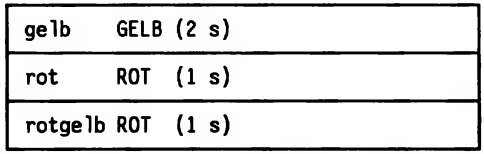


Abb. 4.38 Programmteil Umschaltung Nacht -> Tag

Was jetzt noch fehlt, ist eine Verbindung der Programmteile entsprechend der Aufgabenstellung. Der Wechsel zwischen Tages- und Nachtbetrieb soll nun ja nicht mehr nach einer fest programmierten Anzahl von Schleifendurchläufen erfolgen, sondern in Abhängigkeit vom Tageslicht gesteuert werden. Am Interface ist der Eingabekanal 7 noch frei. Wir schließen dort einen Optosensor an und richten seine Öffnung gegen ein helles Fenster oder eine Lichtquelle im Zimmer. Immer wenn wir mit der Hand die Sensoröffnung abschatten, wird der Eingabekanal 7 aktiviert, was unschwer am Aufleuchten der grünen Leuchtdiode zu erkennen ist. Mit diesem "Verdunkeln" können wir somit die Tages- und Nachtperioden für unsere Aufgabenstellung simulieren.

Die Vorgehensweise bei der Softwareerstellung, bei der kleine, einfache Programmteile für grundlegende Einzelfunktionen zu einem komplexen Gesamtprogramm verbunden werden, nennt man BOTTOM-UP-Verfahren. Dieses Verfahren bietet sich hier aus zwei Gründen an: Erstens haben wir die Einzelmodule "Tagesbetrieb", "Blinken" und "Umschaltung: Nacht -> Tag" bereits zur Verfügung. Sie müssen nur noch zu einem funktionsfähigen Gesamtprogramm verknüpft werden. Zweitens wird dadurch vor allem für die Schüler der Blick auf das Wesentliche frei. Dies soll im folgenden Struktogramm (Abb. 4.39) verdeutlicht werden, in dem die bereits vorliegenden Programmteile lediglich als einfache Rechtecke dargestellt sind.

Ohne sich die Programmierung der Einzelmodule überlegen zu müssen, kann so recht schnell die Strukturierung des Gesamtprogramms erarbeitet werden: Der Tagesbetrieb soll solange wiederholt werden, bis die Aktivierung von Sensor 7 einbrechende Dunkelheit meldet. Danach blinkt die Ampel der Nebenstraße, bis die wiederkehrende Tageshelligkeit den Eingabekanal 7 inaktiviert. Nun muß nur noch die korrekte Umschaltung auf den Tagesbetrieb erfolgen und der Ablauf kann von Neuem beginnen.

Sind sich die Schüler über die Grundstruktur des Gesamtprogramms im klaren, so können jetzt die Einzelmodule durch die konkreten Anweisungen ersetzt werden. Das dadurch entstehende Struktogramm ist durch die dreifache Schachtelung der Schleifen derart komplex, daß seine direkte Erarbeitung mit den Schülern sicherlich fraglich gewesen wäre.

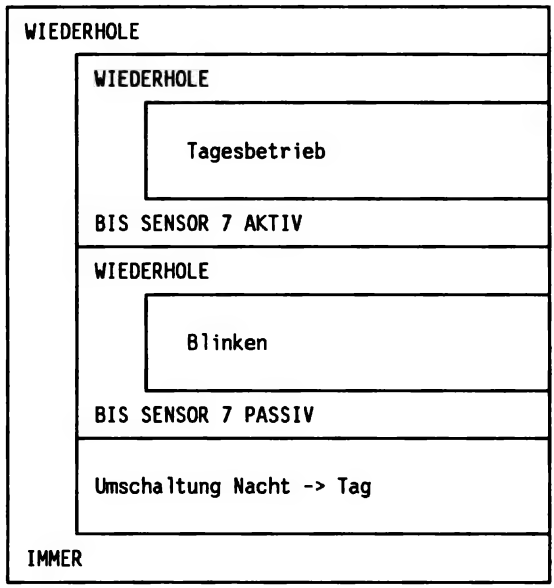


Abb. 4.39 Grundstruktur AMPEL4

Für die Programmierung in LEGO Lines bietet es sich an, auf das Programm AMPEL3 als Grundlage zurückzugreifen. Die beiden WIEDERHOLEn...IMMER-Strukturanweisungen müssen lediglich in die Anweisung WIEDERHOLE...BIS für die Abfrage des Sensors 7 abgeändert werden und die erste Steueranweisung "grün ROT" im Programmteil für den Tagesbetrieb muß ebenfalls in eine WIEDERHOLE...BIS-Abfrage des Sensors 6 integriert werden.

Soll diese Ampelsteuerung auf der Modellebene etwas realistischer dargestellt werden, so benötigt man hierzu zwei zusätzliche LEGO Modellampeln. Da die beiden Ampeln einer Straßenführung völlig parallel geschaltet werden, kann die zweite Ampel für die Hauptverkehrsstraße ebenfalls an die Ausgabekanäle 3 bis 5 und die zusätzliche Ampel für die Nebenstraße an die Kanäle 0 bis 2 angeschlossen werden. Die Stecker der Verbindungskabel bieten dafür genügend Anschlußmöglichkeiten. Natürlich müssen dann auch auf beiden zur Kreuzung hinführenden Spuren der Nebenstraße entsprechende Sensoren angebracht werden.

```
WIEDERHOLE
    WIEDERHOLE
        WIEDERHOLE
            grün    ROT
        BIS SENSOR 6 AKTIV
        grün    ROT      (3 s)
        gelb    ROTGELB  (1 s)
        rot     ROTGELB  (0,2 s)
        rot     GRÜN     (3 s)
        rotgelb GELB     (1 s)
        rotgelb ROT      (0,2 s)
    BIS SENSOR 7 AKTIV
    WIEDERHOLE
        GELB ein für 0,5 s
        GELB aus für 0,5 s
    BIS SENSOR 7 PASSIV
    gelb    GELB (2 s)
    rot     ROT  (1 s)
    rotgelb ROT  (1 s)
IMMER
```

Abb. 4.40 Struktogramm AMPEL4

Auch dies ist kein Problem. Mit einem weiteren Optosensor und einem Leuchtstein wird eine zweite Lichtschranke gebaut. Der Leuchtstein wird parallel zum Leuchtstein der anderen Lichtschranke an den Permanentausgang angeschlossen, der Sensor kommt ebenfalls parallel an den Eingangskanal 6. Damit ist auf der Hardwareebene eine ODER-Verknüpfung realisiert: Unabhängig davon, von welcher Seite sich ein Fahrzeug auf der Nebenstraße nähert, ob die eine **oder** die andere Lichtschranke

unterbrochen wird, kommt es zur Aktivierung des Eingabekanals und damit zur Phasenumschaltung der Ampelanlage.

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
WIEDERHOLE									
WIEDERHOLE									
grün ROT	*	*	1	0	0	0	0	1	
BIS	*	1							
grün ROT	*	*	1	0	0	0	0	1	3,0
gelb ROTGELB	*	*	0	1	0	0	1	1	1,0
rot ROTGELB	*	*	0	0	1	0	1	1	0,2
rot GRÜN	*	*	0	0	1	1	0	0	3,0
rotgelb GELB	*	*	0	1	1	0	1	0	1,0
rotgelb ROT	*	*	0	1	1	0	0	1	0,2
BIS	1	*							
WIEDERHOLE									
GELB ein	*	*	0	0	0	0	1	0	0,5
GELB aus	*	*	0	0	0	0	0	0	0,5
BIS	0	*							
GELB gelb	*	*	0	1	0	0	1	0	2,0
ROT rot	*	*	0	0	1	0	0	1	1,0
ROTGELB rot	*	*	0	1	1	0	0	1	1,0
IMMER									

Abb. 4.41 Programmraster AMPEL4

Aufgabe: Das Programm AMPEL4 hat noch einen Schwachpunkt. Die Umschaltung auf den Nachtbetrieb erfolgt nur am Ende einer Grünphase für den Querverkehr. Solange auf der Nebenstraße kein Auto naht, kann

nicht umgeschaltet werden, da die zuständige Sensorabfrage im Programm nicht erreicht wird. Ändern Sie deshalb das Programm durch Verwendung der Strukturanweisung WENN...ENDE WENN so ab, daß eine Anforderung zur Umschaltung in den Nachtbetrieb immer erkannt wird. Unser Lösungsvorschlag befindet sich als AMPEL5 auf der Diskette. Er enthält außerdem eine Möglichkeit, die Länge der Grünphase für die Nebenstraße vom dortigen Verkehrsaufkommen abhängig zu machen.

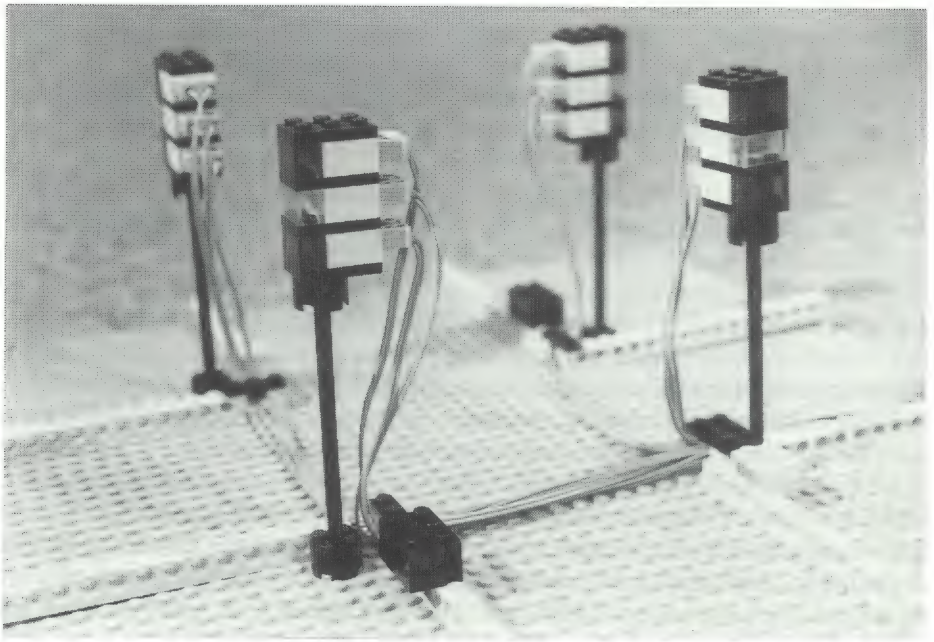


Abb. 4.42 Modell der Ampelanlage

4.4 Projekt: Bahnübergang

4.4.1 Die Ampel am Bahnübergang

Mit dem im folgenden dargestellten Projekt bleiben wir weiterhin beim Thema "Verkehr". Trotz der Strecken- und Fahrplanausdünnungen der Deutschen Bundesbahn gehören beschrankte Bahnübergänge in der Regel nach wie vor zum Verkehrsalltag. Nur wenige Schüler dürften jedoch noch Bahnübergänge mit Schrankenwärterhäuschen und manueller Schrankenbedienung aus der eigenen Anschauung kennen. Heute senken und heben sich Bahnschranken wie von selbst; längst haben moderne Steuerungsanlagen diese verantwortungsvolle Tätigkeit übernommen. Im Unterricht lassen sich diese Bahnübergänge mit dem LEGO Technic Control System leicht nachbilden. Die einzelnen Steuerungsprozesse können so den Schülern auf einfache Weise transparent gemacht werden.

In der Regel sind Bahnübergänge zusätzlich durch Lichtzeichenanlagen geschützt. Schrankenlose Übergänge begnügen sich mit einem blinkenden Rotlicht. An beschrankten Überwegen findet sich vom einfachen Blinklicht über eine Zweiphasenampel mit gelbem und rotem Licht bis zur gewöhnlichen Verkehrsampel ein breites Spektrum an Signalanlagen. Für unser Projekt legen wir uns sinnvollerweise auf eine gewöhnliche Verkehrsampel fest. Zum einen deshalb, weil die übrigen Möglichkeiten vielleicht doch zu einfach wären, zum anderen kann so wieder auf einen bereits vorhandenen Programmteil zurückgegriffen werden.

Die eigentliche Programmierarbeit für die Ampel am Bahnübergang entfällt dadurch. Es bietet sich an, stattdessen mit den Schülern einen vorläufigen Ablaufplan für die Steuerung des Bahnübergangs zu strukturieren. Die einzelnen Ablaufschritte sind schnell aufgezählt und in einem Struktogramm dargestellt: Bei der Annäherung eines Zuges wird die Ampel von Grün über Gelb auf Rot umschalten. Erst dann kann die Schranke gesenkt werden. Wenn der Zug durchgefahren ist, hebt sich die Schranke wieder und die Ampel schaltet auf Grün zurück.

Die Umsetzung des Struktogramms in ein LEGO Lines Programm könnte, was die Steuerung der Ampel angeht, sofort erfolgen. Problematisch wird dagegen der Betrieb der Schranke, da der Schrankenmotor für das Heben und Senken der Schranke in zwei verschiedenen Richtungen

betrieben werden muß. Die Lösung dieses "Problems" soll im nächsten Abschnitt dargestellt werden.

WIEDERHOLE		
	Auto: grün	(10 s)
	Auto: gelb	(1 s)
	Auto: rot	(1 s)
	Auto: rot Schranke ab	(4 s)
	Auto: rot (Zug fährt durch)	(5 s)
	Auto: rot Schranke auf	(4 s)
	Auto: rotgelb	(1 s)
IMMER		

Abb. 4.43 Struktogramm BAHN1

4.4.2 Ansteuerung von Motoren: Die Schranke

Schließt man einen LEGO Gleichstrommotor an eine der von 0 bis 5 bezifferten Buchsen des Interfaces an und aktiviert im Direktmodus den gewählten Ausgabekanal, so wird sich der Motor zwar ein- und ausschalten lassen, er wird sich aber nur in eine Richtung drehen. Will man die Drehrichtung ändern, so muß der Stecker in der Buchse verdreht werden. Durch dieses Umpolen ist zwar eine Laufrichtungsänderung möglich, nachteilig ist aber, daß sie manuell erfolgen muß. Für computergesteuerte Anwendungen ist diese Lösung natürlich nicht praktikabel. Das LEGO Interface bietet deshalb drei weitere Steckbuchsen, die mit den Buchstaben A, B und C bezeichnet sind und die waagerecht zwischen je zwei senkrechten Buchsen liegen. Beim Anschließen des Motors in solch einer Buchse ist ein computergesteuertes Umschalten der Drehrichtung problemlos möglich, wie ein einfaches Programm verdeutlicht.

Ein an der Buchse A des Interfaces angeschlossener Motor wird fünfmal seine Laufrichtung wechseln, da jeweils für eine Sekunde alternierend die Ausgangskanäle 0 und 1 aktiviert werden. Da ein Pol der Buchse A dem

Kanal 0, der andere dem Kanal 1 zugeordnet ist, kommt es zu einem in der Richtung unterschiedlichen Stromfluß, der in einem Gleichstrommotor entgegengesetzte Drehrichtungen bewirkt.

	7	6	5	4	3	2	1	0	
WIEDERHOLE									5
Motor links	*	*	0	0	0	0	0	1	1,0
Motor rechts	*	*	0	0	0	0	1	0	1,0
ENDE WIEDER									
und nun ????	*	*	0	0	0	0	1	1	1,0
	*	*	0	0	0	0	0	0	

Abb. 4.44 Programmraster MOTORTEST

Sofort drängt sich (und nicht nur bei den Schülern) die Frage auf, was denn bei einer gleichzeitigen Aktivierung beider Ausgabekanäle passiert. In der vorletzten Zeile des obigen Rasters ist dieser Fall dargestellt. Gelangt das Programm bei seiner Ausführung an diese Stelle, so bleibt der Motor einfach stehen. Dieses Ergebnis dürfte bei nochmaligem Überdenken nicht verwundern, da ein Elektromotor nur laufen kann, wenn zwischen seinen beiden Polen ein unterschiedliches elektrisches Potential anliegt. Schließt man beide Pole eines Motors jedoch an den Plus- oder an den Minuspol einer Batterie an, so kann er nicht funktionieren. Aus diesem Grund stellt ein gleichzeitiges Aktivieren beider Kanäle für den angeschlossenen Motor auch keine Überlastung oder gar Gefährdung dar.

Mit diesen Kenntnissen kann nun an den Bau einer Schranke gegangen werden. Durch die hohe Drehgeschwindigkeit der Elektromotoren ist eine Getriebeuntersetzung für den Betrieb der Schranke erforderlich. Die LEGO Technic Control Sets enthalten hierzu eine Vielzahl von Getriebeteilen wie Zahnräder, Schnecken und Wellen. Je nach Ausprägung des Unterrichts und Fortschrittsgrad der Schüler im Umgang mit dem LEGO Material wird man als Lehrer mehr oder weniger stark mit Bauanleitungen eingreifen wollen. Da den Sets ohnehin keine Anleitungen für den Aufbau einer Schranke beiliegen, scheint es durchaus angebracht, den Schülern entsprechende Freiräume für das Verwirklichen eigener Kon-

struktionen zuzugestehen. Einem probierenden Vorgehen aus eigenem Antrieb dürften unter Umständen wesentlich wertvollere Erkenntnisse über den Getriebebau entspringen, als in einer geplanten Unterrichtsstunde je vermittelt werden können. In den an der Pädagogischen Hochschule Schwäbisch Gmünd durchgeführten Projekten arbeiteten alle Schülergruppen mit Begeisterung bis zum funktionierenden Ergebnis. Dankbar waren die meisten Gruppen allerdings für eine "Starthilfe" beim Getriebebau. Eine solche Starthilfe wurde der LEGO Bauanleitung für die Förderbandanlage entnommen, wo ebenfalls ein Untersetzungsgetriebe benötigt und mit einem einfachen Schneckengetriebe realisiert wird.

Trotz dieser gemeinsamen Basis werden die Ergebnisse der einzelnen Schülergruppen stark voneinander abweichen. Dies ist jedoch nicht tragisch, da die Programmierung der Schranke sehr einfach sein wird, so daß kein Auseinanderdriften des Unterrichts befürchtet werden muß.

4.4.3 Zeitsteuerung oder Schrittsteuerung?

Gleich nach der Fertigstellung "ihrer" Schranke werden die Schüler sie ans Interface anschließen und über den Computer steuern wollen. Für ein erstes Öffnen und Schließen der Schranke kann das kurze Programm aus dem letzten Abschnitt (Abb. 4.44) verwendet werden; lediglich die Zeiten für das Heben und Senken müssen verändert werden. Aufgrund der verschiedenen Konstruktionen muß jede Gruppe die für ihre Schranke nötige Zeitdauer selbst experimentell ermitteln. Legt man in der Klasse fest, daß Kanal 0 die Schranke senken und Kanal 1 sie heben soll (gegebenenfalls muß der Stecker am Motor oder Interface umgepolt werden), so ist die individuell ermittelte Zeitdauer der einzig unterschiedliche Faktor in der weiteren Arbeit mit den Schrankenmodellen der Klasse.

Die im Programmraster AMPEL1 für die Ampelsteuerung getroffene Zuordnung der Leuchtsteine auf die Kanäle 3 bis 5 soll beibehalten werden. Das bereits erarbeitete Struktogramm BAHN1 ergibt damit das untenstehende Programm (Abb. 4.45).

Anstelle der Fragezeichen muß natürlich die für das jeweilige Schrankenmodell zum Öffnen und Schließen ermittelte Zeitdauer eingetragen

werden. Da der Schrankenarm jeweils um denselben Winkel gedreht wird, erwartet man sicherlich zwei gleiche Werte für das Heben und Senken. Diejenigen Schülergruppen, die das Programm mit experimentell ermittelten gleichen Zeitwerten mehrere Male hintereinander ablaufen lassen, werden aber sehr schnell feststellen, daß bald eine Neujustierung der Schranke notwendig wird. In der Regel wird sich der Schrankenarm weiter senken als heben, so daß er im geöffneten Zustand die Senkrechte nicht mehr erreicht. Den Grund hierfür können die Schüler unschwer selbst nennen: Der Motor muß ja beim Öffnen wesentlich mehr Arbeit verrichten und benötigt deshalb auch mehr Zeit hierfür als beim Schließen. Als Lösungsmöglichkeit bietet es sich zunächst an, die Zeitdauer für das Heben länger zu bemessen. In unzähligen Versuchsläufen sucht nun jede Gruppe die für ihr Schrankenmodell passenden Zeiten zu ermitteln.

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
grün	*	*	1	0	0	0	0	0	2,0
gelb	*	*	0	1	0	0	0	0	1,0
rot	*	*	0	0	1	0	0	0	1,0
rot ab	*	*	0	0	1	0	0	1	?
rot ZUG	*	*	0	0	1	0	0	0	2,0
rot auf	*	*	0	0	1	0	1	0	?
rotgelb	*	*	0	1	1	0	0	0	1,0
IMMER									

Abb. 4.45 Programmraster BAHN1

Leider erweist sich alle Mühe als umsonst. Läßt man das Programm nur häufig genug laufen, kommt die Schranke immer wieder aus dem Tritt. Daran ändert auch die mögliche Auflösung der Zeitintervalle im Zehntelsekundenbereich nichts. Die Ursache ist in der Getriebereibung und in den schlechten Gleichlaufeigenschaften von Gleichstrommotoren zu suchen.

Bevor die Schüler jedoch enttäuscht das Handtuch werfen wollen, kann man als Lehrer eine überzeugende Lösungsmöglichkeit präsentieren. Daß die Problemlösung mit einer Zeitsteuerung nicht funktioniert, ist inzwischen einsichtig geworden. Zudem erweist sich die Steuerung über den Faktor Zeit bei näherem Hinsehen auch als Umweg. Ursächlich handelt es sich ja um die Steuerung einer Bewegung und charakteristisch für eine Bewegung ist der Weg, den ein bewegter Körper zurücklegt. In vielen Sportdisziplinen wird deshalb mit dem Metermaß die erbrachte Leistung gemessen. Niemand käme auf die Idee, den Sieger im Hochsprung oder beim Speerwurf mit der Stoppuhr feststellen zu wollen. Mißt man, wie bei den Laufdisziplinen, mit der Uhr, so kommt sofort die Geschwindigkeit ins Spiel. Sieger wird, wer die 100-Meter-Distanz am schnellsten zurücklegt.

Eine andere Möglichkeit, bei Laufdisziplinen den Sieger zu küren, ist deshalb zumindest theoretisch denkbar: Stellt man bei jedem Läufer fest, wie weit er beispielsweise in 10 Sekunden kommt, so ist derjenige der Schnellste, der in dieser Zeit den größten Weg zurückgelegt hat. Dies ergibt sich direkt aus der Definition der Geschwindigkeit. Einige Weltklasseläufer werden in der angegebenen Zeitspanne die Einhundertmeter-Marke erreichen, viele andere jedoch nicht. Allerdings werden auch die Weiten, die ein und derselbe Läufer an verschiedenen Tagen nach der beschriebenen Methode zurücklegt, entsprechend seiner Tagesform variieren, da er nicht in jedem Durchgang mit exakt der gleichen Geschwindigkeit laufen wird.

Möchte man die bei einer Bewegung zurückgelegte Wegstrecke reproduzierbar mit der Stoppuhr bestimmen, so setzt dies voraus, daß die Bewegung immer mit genau der gleichen Geschwindigkeit abläuft, eine Voraussetzung, die bei unserer Modellschranke aus den verschiedensten Gründen nicht zugrundegelegt werden kann. Deshalb wird hier und bei ähnlichen Problemstellungen eine Zeitsteuerung immer problematisch bleiben. Sehr viel besser und vollkommen zeitunabhängig wäre es, könnte man den bei der Bewegung der Schranke zurückgelegten Weg mitverfolgen und beim Erreichen einer bestimmten Marke die Bewegung stoppen.

Da es sich bei der Bewegung der Schranke und des vorgeschalteten Getriebes um Drehbewegungen handelt, wäre für eine Wegbestimmung die Feststellung der Umdrehungsanzahl des Motors oder einer bestimmten Getriebewelle völlig ausreichend. Genau zu diesem Zweck enthält das

LEGO Technic Control Set für jeden Optosensor eine Segmentscheibe. Diese weiße Segmentscheibe hat auf einer Seite vier, auf der anderen Seite acht schwarze Segmente kuchenstückartig aufgezeichnet, die vom Optosensor registriert werden können.

Für ein erstes Kennenlernen schieben wir die Scheibe auf eine Getriebeachse und stecken diese Achse dann so durch die durchgehende Bohrung des Sensors, daß eine Seite der Scheibe dicht vor die kleine ovale Sensoröffnung zu liegen kommt. Den Sensor schließen wir an die Eingangsbuchse 6 des betriebsbereiten Interfaces an und drehen die Scheibe am Sensor. Die an der Sensoröffnung vorbeibewegten Hell-Dunkel-Wechsel werden am Interface durch das Aufleuchten und Erlöschen der grünen Leuchtdiode angezeigt.

Der Sensor ist für das Erkennen der Segmente auf keinerlei Fremdlicht angewiesen. Möglich wird dies durch eine eingebaute Infrarot-Leuchtdiode, deren für uns Menschen unsichtbares Licht von den Segmenten unterschiedlich stark reflektiert wird. Der eingebaute Fototransistor ist auch im Infrarotbereich empfindlich, so daß er die gespiegelten Strahlen erkennen und korrekt auswerten kann. Der Optobaustein arbeitet zusammen mit der Segmentscheibe als Reflexsensor. Für eine zuverlässig Auswertung der einzelnen Impulse durch den Sensor und das Interface darf eine maximale Hell-Dunkel-Frequenz nicht überschritten werden. Die Seite mit den acht schmalen Segmenten ist zwar für eine genauere Positionierung besser geeignet, sie kann jedoch nur für Drehzahlen bis maximal 140 U/min eingesetzt werden, die andere Seite mit den vier breiten Abschnitten bis zur doppelten Drehzahl.

Wenn wir nun die Segmentscheibe auf eine Getriebeachse unserer Schranke aufstecken und durch einen Sensor auswerten lassen, kann die notwendige Bestimmung der Anzahl der Getriebeumdrehungen für das Öffnen und Schließen der Schranke durch das Abzählen der Hell-Dunkel-Wechsel erfolgen. Die Anzahl dieser Wechsel wird jetzt für das Heben und Senken genau gleich und vor allen Dingen immer konstant sein. Mit der Kombination Sensor - Segmentscheibe ist nun eine direkte Bestimmung einer Bewegung möglich geworden, indem der zurückzulegende Weg durch die Segmente in kleine Schritte unterteilt wird, die ihrerseits erfaßt und ausgewertet werden können.

Da die Steuerung nun nicht mehr über die Zeit, sondern über kleine Schritte erfolgt, kann man hierbei auch von einer Schrittsteuerung sprechen. Diese Steuerungsmethode wird derart häufig eingesetzt, daß es hierfür sogar spezielle Motoren, sogenannte Schrittmotoren gibt. Diese Motoren benötigen zu ihrem Betrieb nicht eine konstante Spannung, sondern einzelne Spannungsimpulse. Jeder Spannungsimpuls dreht den Motor um einen genau festgelegten kleinen Schritt in der Größenordnung von wenigen Winkelgraden oder gar nur um Bruchteile von Graden weiter. Häufig werden solche Motoren in Computersteuerungen eingesetzt. Dies hat den Vorteil, daß der Computer durch einfaches Mitzählen der Spannungsimpulse, die er an den Motor sendet, immer genauestens über die Position der angetriebenen Einheit informiert ist.

Dieses Prinzip werden wir nun auch an unserer Schranke anwenden. Zwar können wir keinen Schrittmotor einsetzten, aber der Computer kann ja die Impulse der auf eine Getriebeachse gesteckten Segmentscheibe zählen und beim Erreichen einer vorher festgelegten Schrittzahl die Bewegung stoppen. Voraussetzung hierfür ist nur noch eine Strukturanweisung in LEGO Lines, die den Computer mit den geforderten Fähigkeiten ausstattet. Die Anweisung ZÄHLE leistet genau das Gewünschte und ist wie alle anderen Strukturanweisungen einfach anzuwenden, wovon wir uns gleich in einem ersten Beispiel überzeugen werden.

Dazu stellen wir uns die Aufgabe, daß das Rotlicht unserer Ampel solange brennen soll, bis der Sensor zehn Hell-Dunkel-Wechsel der Segmentscheibe registriert hat; danach soll auf Grün umgeschaltet werden. Für dieses kleine Beispielprogramm brauchen wir das momentan im Speicher befindliche Programm BAHN1 nicht löschen. LEGO Lines bietet die Möglichkeit, mehrere Programme unabhängig voneinander im Speicher zu halten, solange alle diese Programme zusammengenommen weniger als 40 Zeilen lang sind und zwischen verschiedenen Programmen mindestens eine Zeile völlig leer bleibt.

Wir wollen diese Möglichkeit nutzen, da das aktuelle Programm BAHN1 später noch benötigt wird. Lassen wir also nach dem existierenden Programm mindestens eine Zeile völlig leer und beginnen dann mit unserer Aufgabe. Über den Ausgangskanal 3 wird das rote Licht der angeschlossenen Ampel eingeschaltet. In die folgende Zeile kommt die ZÄHLE-Anweisung (Commodore: ZAEHLE). Wieder wird der eingegebene Befehl nach dem letzten Tastendruck in Großbuchstaben um-

gesetzt und das Raster der Ausgabekanäle verschwindet; lediglich auf die Eingabekanäle 6 und 7 kann noch zugegriffen werden.

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
grün	*	*	1	0	0	0	0	0	2,0
gelb	*	*	0	1	0	0	0	0	1,0
rot	*	*	0	0	1	0	0	0	1,0
rot ab	*	*	0	0	1	0	0	1	?
rot ZUG	*	*	0	0	1	0	0	0	2,0
rot auf	*	*	0	0	1	0	1	0	?
rotgelb	*	*	0	1	1	0	0	0	1,0
IMMER									
	*	*							
rot	*	*	0	0	1	0	0	0	
ZÄHLE	*	0							10
grün	*	*	1	0	0	0	0	0	2,0

Abb. 4.46 Zwei Programme im LEGO Lines Raster

Da LEGO Lines nur die Impulse eines Eingabekanals "zählen" kann und wir den Sensor an Kanal 6 angeschlossen haben, muß jetzt der Kanal 7 ausgeblendet werden. Drücken Sie deshalb solange auf die Taste <7>, bis im zugehörigen Rasterfeld eine Schraffur erscheint. Der Inhalt des Eingaberasters für Kanal 6 ist hingegen völlig gleichgültig, hier kann eine 1 oder eine 0 erscheinen. Da wir nur die Impulswechsel zählen, ist es schließlich egal, ob wir uns hierzu der Hell-Dunkel- oder der Dunkel-Hell-Wechsel bedienen. Ins Wertefeld schreiben wir die Zahl 10, die nun keine Zeitdauer darstellt, sondern die Anzahl der zu zählenden Impulse angibt. In die dritte Zeile kommt die Anweisung, das grüne Licht für 2 Sekunden einzuschalten. Im Speicher befinden sich nun zwei Programme, die natürlich nur zum Teil auf dem Bildschirm sichtbar sind.

Zum Starten des zweiten Programms muß sich der Marker auf einer der drei Zeilen dieses Programms befinden. Nach dem Programmstart mit **<F9>** (Commodore: **<f7>**) springt der Marker automatisch in die erste Programmzeile des zweiten Programms und Kanal 3 wird aktiviert. Beachten Sie, daß in dieser Zeile keine Ausführungsdauer angegeben wurde. Dies ist nicht nötig und auch nicht sinnvoll, da die Zustände der Ausgabekanäle solange beibehalten werden, bis LEGO Lines auf eine neue Ausgabeanweisung trifft. In der nächsten Zeile steht jedoch die ZÄHLE-Anweisung und dort verharrt der Marker, bis die angegebenen zehn Impulse vom Sensor am Interface eingetroffen sind. Erst dann wird in der letzten Zeile das grüne Licht der Ampel eingeschaltet.

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
grün	*	*	1	0	0	0	0	0	2,0
gelb	*	*	0	1	0	0	0	0	1,0
rot	*	*	0	0	1	0	0	0	1,0
rot ab	*	*	0	0	1	0	0	1	
ZÄHLE	*	0							?
rot ZUG	*	*	0	0	1	0	0	0	2,0
rot auf	*	*	0	0	1	0	1	0	
ZÄHLE	*	0							?
rotgelb	*	*	0	1	1	0	0	0	1,0
IMMER									

Abb. 4.47 Programmraster BAHN2

Einer Verbesserung des Programms BAHN1 steht nun nichts mehr im Wege. Zuvor muß lediglich die Segmentscheibe auf eine Getriebewelle aufgesteckt und der Sensor hinzugefügt werden. Wie dies zu geschehen hat, hängt von der jeweiligen Konstruktion des Schrankenmodells ab. Unter Umständen muß eine Getriebewelle durch eine längere ersetzt werden, um die Segmentscheibe zusätzlich aufstecken zu können. Es empfiehlt sich, hierzu eine Getriebewelle zu wählen, welche im Betrieb mit mittlerer Drehzahl dreht, um einerseits die genannten Grenzen nicht

zu überschreiten und um andererseits eine möglichst große Auflösung der Schritte zu erreichen.

Auf dem Bildschirm bedarf es nur weniger Änderungen. Nach den Zeilen, die das Schließen und Öffnen der Schranke bewirken, fügen wir jeweils eine ZÄHLE-Anweisung ein. Die zuvor experimentell ermittelten Zeitwerte für das Heben und Senken der Schranke müssen gelöscht werden, da sonst das Zählen der Sensorimpulse erst nach der angegebenen Zeit erfolgt. Welcher Zählwert im Wertefeld der ZÄHLE-Anweisung angegeben werden muß, hängt wieder von der jeweiligen Konstruktion ab; er kann ebenfalls experimentell ermittelt werden. Läuft die Achse, auf welcher die Segmentscheibe angebracht wurde, nicht zu schnell, so kann man die Schranke im Direktmodus senken bzw. heben und die dabei erzeugten Blinkimpulse der grünen Leuchtdiode auf dem Interface zählen. Das dreizeilige Beispielprogramm löschen wir am besten wieder, so daß nur noch das obige Programm BAHN2 im Speicher ist.

Anstatt der Fragezeichen trägt man natürlich die für das jeweilige Modell ermittelte Impulsanzahl ein. Die Schrittsteuerung sorgt nun dafür, daß selbst nach vielmaligem Programmablauf die Schranke immer noch korrekt öffnet und schließt.

4.4.4 Automatisierung des Bahnübergangs

Im Hinblick auf eine sicherlich wünschenswerte Realitätsnähe hat unsere jetzige Lösung leider noch einen Schwachpunkt. Züge fahren wohl kaum in solch konstanten Zeitabschnitten, wie dies das jetzige Programm impliziert. Auch ist man längst weg von einer manuellen Steuerung eines Bahnübergangs. In der Regel löst ein herannahender Zug selbst das Senken und Heben der Schranke aus.

Mit dem LEGO Material läßt sich solch eine wirklichkeitsnahe Darstellung leicht nachbilden. In einem ersten Ansatz soll die Bahnschranke beim Herannahen eines Zuges automatisch geschlossen und nach Ablauf einer gewissen festgelegten Zeitspanne wieder geöffnet werden. Die Ankunft eines Zuges wird, genau wie beim Ampel-Projekt, von einer Lichtschranke gemeldet. Der Optosensor dieser Lichtschranke wird an Kanal 7, der zugehörige Leuchtstein an den Permanentausgang angeschlossen. Aus der Überlegung, daß die Autofahrer solange grünes

Licht erhalten sollen, bis der Sensor 7 aktiv wird, kann sofort das folgende Struktogramm (Abb. 4.48) erarbeitet werden.

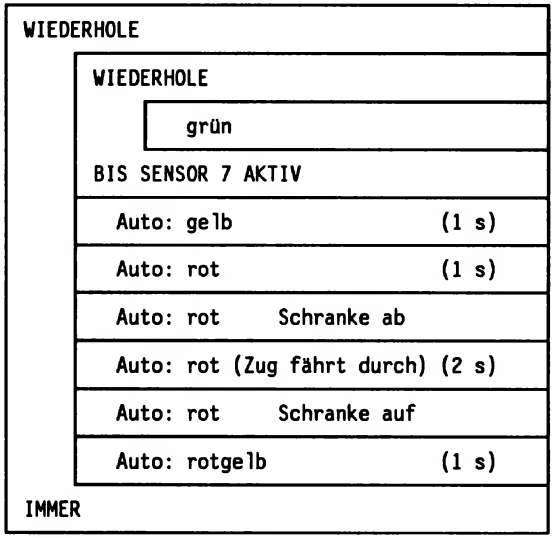


Abb. 4.48 Struktogramm BAHN3

Die Umsetzung in ein LEGO Lines Programm ist einfach, lediglich zwei Zeilen müssen in das Programm BAHN2 eingefügt und die Angabe der Zeitdauer in der Zeile "grün" gelöscht werden.

Problematisch ist jetzt nur noch die Vorgabe einer festen Zeitdauer, die für das Durchfahren des Zuges vorgegeben wird. Durch die Verwendung einer einzelnen Lichtschranke darf sich in unserer jetzigen Lösung ein Zug außerdem nur von einer Seite nähern. Für eine prinzipielle Darstellung des Steuerungsvorganges mag dies genügen, für Perfektionisten, die auf einen weiteren Optosensor zugreifen können, gibt es jedoch noch eine realistischere Lösung.

Mit dem dritten Sensor und einem weiteren Leuchtstein installieren wir eine zweite Lichtschranke auf der anderen Seite der Schranke. Der Leuchtstein wird ebenfalls aus dem Permanentausgang versorgt, der dritte Sensor wird parallel zum zweiten an den Eingabekanal 7 angeschlossen. Damit haben wir, ohne Veränderung des Programms, bereits eine entscheidende Verbesserung erreicht: Durch das Parallelschalten beider

Lichtschrankensensoren wurde hardwaremäßig eine ODER-Schaltung erzeugt, jetzt wird ein Herannahen eines Zuges aus jeder Richtung erkannt.

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
grün	*	*	1	0	0	0	0	0	
WIEDERHOLE									
BIS	1	*							
gelb	*	*	0	1	0	0	0	0	1,0
rot	*	*	0	0	1	0	0	0	1,0
rot ab	*	*	0	0	1	0	0	1	
ZÄHLE	*	0							?
rot ZUG	*	*	0	0	1	0	0	0	2,0
rot auf	*	*	0	0	1	0	1	0	
ZÄHLE	*	0							?
rotgelb	*	*	0	1	1	0	0	0	1,0
IMMER									

Abb. 4.49 Programmraster BAHN3

Da nun an beiden Seiten der Schranke ein Sensor am Gleiskörper vorhanden ist, können wir auf eine sicherheitstechnisch problematische zeitgesteuerte Öffnung des Bahnübergangs verzichten und das Heben der Schranke vom durchgefahrenen Zug initiieren lassen. Hierzu bedarf es nur einer kleinen Ergänzung des bereits vorhandenen Programms.

Die Schranke bleibt nun solange geschlossen, bis der Zug nach der Durchfahrt am Bahnübergang die zweite Lichtschranke passiert hat. Ausgehend von dem rein sequentiellen und einfachen Programm BAHN1 konnte schrittweise eine durchaus komplexe Aufgabenstellung realisiert werden, die zudem dem Anspruch einer möglichst wirklichkeitsgetreuen Abbildung der Realität auf die Modellebene weitgehend gerecht wird.

Dieses schrittweise Fortentwickeln von Lösungsansätzen wird durch die hohe Transparenz von LEGO Lines Programmen stark unterstützt. Eine sinnvolle Verwendung des Textfeldes für Kommentare und die ausgeprägte Gliederung durch Strukturanweisungen machen vorliegende Programme für Schüler leicht lesbar und verständlich. Eine gezielte Änderung von Programmen wird zusätzlich durch die einfachen Editiermöglichkeiten unterstützt.

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
grün	*	*	1	0	0	0	0	0	
WIEDERHOLE									
BIS	1	*							
gelb	*	*	0	1	0	0	0	0	1,0
rot	*	*	0	0	1	0	0	0	1,0
rot ab	*	*	0	0	1	0	0	1	
ZÄHLE	*	0							?
rot ZUG	*	*	0	0	1	0	0	0	
WIEDERHOLE									
BIS	1	*							
rot auf	*	*	0	0	1	0	1	0	
ZÄHLE	*	0							?
rotgelb	*	*	0	1	1	0	0	0	1,0
IMMER									

Abb. 4.50 Programmraster BAHN4

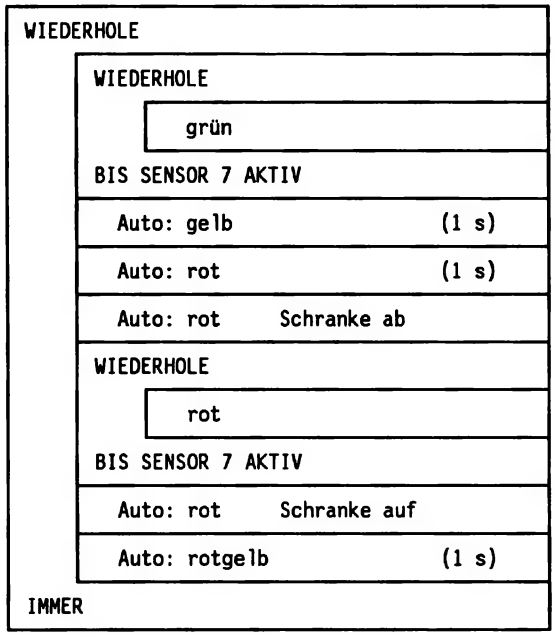


Abb. 4.51 Struktogramm BAHN4

Zwischenbemerkung

Die beiden ersten Projekte "Verkehrsampel" und "Bahnübergang" sind in ihrem kleinschrittigen Aufbau vor allem für eine Einführung in die Arbeit mit LEGO Lines konzipiert. Ziel war, zunächst den Umgang mit der graphischen Benutzeroberfläche einzuüben und die verschiedenen Strukturanweisungen in ihrem richtigen Gebrauch vorzustellen. Daß schon bei diesen einführenden Projekten bereits durchaus anspruchsvolle Aufgabenstellungen bearbeitet werden konnten, unterstreicht deutlich die leichte Erlernbarkeit und Leistungsfähigkeit von LEGO Lines. Bei den folgenden Projekten wird darauf aufbauend von einer gewissen Fertigkeit im Umgang mit LEGO Lines ausgegangen. Für eine Konzentration auf die jeweils anstehende Problematik soll nicht mehr jeder notwendige Tastendruck explizit angegeben werden. Im Zweifelsfalle finden Sie im Anhang des Buchs die Belegung der Funktionstasten beider Rechnertypen und ein Verzeichnis der verfügbaren Strukturanweisungen.

4.5 Projekt: Sortieranlage

4.5.1 Aufbau der Sortieranlage

Für den Bau der Sortieranlage dient die Aufbauanleitung 1090 D. Die Bauanleitungen für die vorgeschlagenen Modelle der LEGO Technic Control Sets sind konsequent nach demselben Schema aufgebaut: Auf dem Titelblatt ist das fertige Modell zusammen mit einer realen Verteileranlage abgebildet, der Bezug zur technischen Wirklichkeit wird so augenfällig dargestellt. Die nächste Seite zeigt die für den Bau benötigten Teile und deren Anzahl, womit eine Vorplanung des Zusammenbaus durch die Bereitstellung der Einzelteile möglich wird. Der eigentliche Zusammenbau ist durch die Abbildung sehr vieler aufeinanderfolgender Bauphasen kleinschrittig und detailliert dargestellt.

Der Aufbau der Sortieranlage ist mit dieser ausgezeichneten Anleitung problemlos zu bewältigen. Versierte Schülergruppen werden diese Aufgabe in etwas mehr als dreißig Minuten erledigen; diese Zeitangabe ist vor allem davon abhängig, ob für einen erstmaligen Bau noch die 94 einzeln gelieferten Kettenglieder zu den beiden Förderbändern zusammengesetzt werden müssen.

Während der Bauphase wird aufgrund der guten Dokumentation die Mithilfe des Lehrers kaum erforderlich sein. Es ist jedoch sicherlich sinnvoll und notwendig, die Schüler zuvor auf das genaue Einhalten der Anleitung hinzuweisen. Die folgende Programmierung der Anlagen gestaltet sich dann einheitlicher und das beabsichtigte spätere Zusammenkoppeln mehrerer Modelle zu einer komplexen Sortieranlage wird dadurch erst ermöglicht.

In jedem Fall ist ein Lehrerhinweis im Hinblick auf die zu verwendenden Achsen nötig. In der Bauanleitung steht neben jeder Achse eine Zahl, welche deren Länge in "Knöpfen" angibt. Die Länge der Achse kann somit im Vergleich mit einem LEGO Baustein ermittelt werden. Es ist dabei wichtig, daß keine längeren Achsen als angegeben verwendet werden. Obwohl dies von der Konstruktion her durchaus möglich wäre, kann es später beim Betrieb des Modells an den überstehenden Achsabschnitten zu stark erhöhter Reibung kommen, welche die Funktion entscheidend beeinträchtigen kann!

In der Endphase des Baus sollten die Schüler vorerst noch auf das Anbringen der Optosensoren und Lampenbausteine verzichten. Die Verkabelung der beiden Antriebsmotoren mit dem Interface wird ganz am Ende der Bauanleitung deutlich herausgestellt. Für eine gemeinsame Ausgangsbasis empfiehlt es sich, die Kabelstecker am Motor oder am Interface so zu polen, daß einheitliche Laufrichtungen erreicht werden. Weiterhin ist es empfehlenswert, die beiden Bänder der Anlage eindeutig zu benennen: Das obere Band, auf welches das Sortiergut aufgelegt wird, nennen wir das Förderband (FB). Das zweite Band, welches die Verteilerfunktion übernimmt, ist das Verteilerband (VB).

Der Motor des Förderbandes soll so an das Interface angeschlossen werden, daß die Aktivierung des Ausgabekanals 2 das Band vorwärts, also in Richtung Verteilerband, laufen läßt. Über den Ausgabekanal 3 läßt sich dann das Förderband rückwärts betreiben. Die Laufrichtung des Verteilerbandes kann, vom Förderband aus gesehen, nach rechts oder links gewählt werden. Der Anschluß des betreffenden Motors wird so verabredet, daß Kanal 4 den Rechts- und Kanal 5 den Linkslauf bewirkt.

Ist die Sortieranlage nun soweit aufgebaut und angeschlossen, möchte man sie sicherlich gleich ausprobieren. In diesem Zusammenhang stellt sich bald die Frage nach einem geeigneten Transport- bzw. Sortiergut. Im LEGO Handbuch wird auf LEGO Steine und Platten verwiesen, man stellt jedoch relativ bald fest, daß dieses Material vor allem für ein später angestrebtes Sortieren nicht unbedingt geeignet ist. Das LEGO Material ist zu leicht und zu glatt, so daß es vom Band nicht einwandfrei erfaßt und transportiert wird. Häufig verklemmt es sich zudem beim Übergang vom Transport- auf das Verteilerband. Diese Probleme lassen sich durch die Verwendung von anderem Material zwar auch nicht völlig beseitigen, aber doch etwas mildern. Relativ gut geeignet scheinen Radiergummis zu sein. Wir haben weiße Läufer-Plast-Radiergummis verwendet und in kleine Quader mit einem Querschnitt von 10×10 mm^2 zugeschnitten. Für eine erste Arbeitsphase ist eine Quaderlänge von 24 mm zu empfehlen.

4.5.2 Einfache Programmierübungen

Als Einstieg in die Programmierung der Verteileranlage kann die Analyse eines vom Lehrer vorgegebenen Programms (Abb. 4.52) dienen.

Die Schüler erkennen unschwer, daß aufgelegtes Sortiergut vom Förderband vorwärts transportiert und vom Verteilerband abwechselnd nach rechts und links geleitet wird. Das Programm arbeitet zeitgesteuert, die Bänder sind abwechselnd in Betrieb. Während das Förderband stillsteht, kann wieder ein Radiergummi aufgelegt werden.

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
FB vor	*	*	0	0	1	0	0	0	6,0
VB rechts	*	*	0	1	0	0	0	0	5,0
FB vor	*	*	0	0	1	0	0	0	6,0
VB links	*	*	1	0	0	0	0	0	5,0
IMMER									

Abb. 4.52 Programmraster TRANS1

Die praktische Erprobung zeigt die Richtigkeit der gemachten Überlegungen, weist aber auch auf eine Schwachstelle des Programmes hin. Die Übergabe des Sortiergutes vom Förder- zum Verteilerband ist unbefriedigend, da der Radiergummi auf das noch stillstehende Verteilerband geschoben wird.

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
FB vor	*	*	0	0	1	0	0	0	4,0
FB vor VB re	*	*	0	1	1	0	0	0	2,0
VB rechts	*	*	0	1	0	0	0	0	4,0
FB vor	*	*	0	0	1	0	0	0	4,0
FB vor VB li	*	*	1	0	1	0	0	0	2,0
VB links	*	*	1	0	0	0	0	0	4,0
IMMER									

Abb. 4.53 Programmraster TRANS2

Wenn das Verteilerband kurz zuvor anlaufen würde, könnte die Übernahme sicherlich besser funktionieren. Aus dieser Anforderung resultiert die verbesserte Programmversion TRANS2. Nun klappt die Übergabe des Sortierguts vom Förder- zum Verteilerband recht gut. Als störend erweist sich jedoch bald, daß die Radiergummis immer im Takt aufgelegt werden müssen. Im Zuge der Humanisierung von Fließbandarbeiten ist es sicherlich wünschenswert, wenn der Mensch einen größeren Einfluß auf die Maschine ausüben kann.

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
Lampe ein	*	*	0	0	0	0	0	1	0,1
WIEDERHOLE									
BIS	1	*							
FB vor	*	*	0	0	1	0	0	0	4,0
FB vor VB re	*	*	0	1	1	0	0	0	2,0
VB rechts	*	*	0	1	0	0	0	0	3,0
Lampe ein	*	*	0	0	0	0	0	1	0,1
WIEDERHOLE									
BIS	1	*							
FB vor	*	*	0	0	1	0	0	0	4,0
FB vor VB li	*	*	1	0	1	0	0	0	2,0
VB links	*	*	1	0	0	0	0	0	3,0
IMMER									

Abb. 4.54 Programmraster TRANS3

Auf der Modellebene läßt sich dies an unserer Sortieranlage problemlos mit einer Lichtschranke realisieren: Auf der Schüttöffnung des Förderbands werden dazu jeweils gegenüberstehend ein Leuchtstein und ein Optosensor angebracht. Der Sensor wird mit dem Eingabekanal 7 verbunden, der Leuchtstein erhält seine Energie vom Ausgabekanal 0. Jetzt kann jedes Auflegen eines Radiergummis auf das Förderband durch

die Unterbrechung der Lichtschranke erkannt werden, der jeweilige Förderbeginn wird somit durch ein Aktivieren von Eingang 7 gestartet.

Nun wartet die Anlage, bis ein Radiergummi auf das Förderband gelegt wird. Nach einem erfolgten Transportvorgang leuchtet die Lampe an der Lichtschranke auf, signalisiert damit die erneute Förderbereitschaft und wartet auf den nächsten Radiergummi. Unsere Förderanlage ist durch die Auswertung eines Sensorsignals flexibel geworden; ein echtes Sortieren ist jedoch noch nicht möglich.

4.5.3 Programmgesteuertes Sortieren

Für ein selbständiges Sortieren der Maschine muß ebenfalls ein Sensorsignal ausgewertet werden. Durch die Verwendung optischer Sensoren können lediglich optisch zu erfassende Unterscheidungskriterien für eine Sortierung in Betracht kommen, am einfachsten ist hier die Unterscheidung nach der Größe des Sortiergutes. Zu diesem Zweck halbieren wir einige Radiergummiquader der Länge nach, so daß sie nur noch einen Querschnitt von 10×5 mm^2 aufweisen.

Dann wird der zweite Sensor zusammen mit einem Leuchstein, wie in der Bauanleitung angegeben, am Ende des Förderbandes als Lichtschranke angebracht. Der Sensor wird mit Kanal 6, der Leuchtstein mit Kanal 1 verbunden. Aufgrund der geometrischen Anordnung werden die flachen Radiergummis unter der Lichtschranke hindurchtransportiert, ohne den Lichtstrahl zu unterbrechen, während ein Radiergummi mit hohem Querschnitt vom Sensor erfaßt wird.

Da der Sortiervorgang weiterhin durch die Aktivierung von Eingang 7 gestartet werden soll, ergibt sich eine recht knifflige Problemstellung. Eine mögliche Lösung besteht darin, den Sensor 6 genau in dem Moment abzufragen, in welchem ein zuvor am Bandanfang aufgelegter Radiergummi am Förderbandende angekommen sein muß. Meldet der Eingang 6 eine Unterbrechung des Lichtstrahls, so kann dies nur durch einen hohen Radiergummi verursacht werden, der auf dem Verteilerband nach rechts transportiert wird. Signalisiert Sensor 6, daß keine Unterbrechung der Lichtschranke erfolgt ist, so wird der dann auf dem Band befindliche flache Radiergummi nach links transportiert. Der Programmablauf wird zunächst in einem Struktogramm dargestellt.

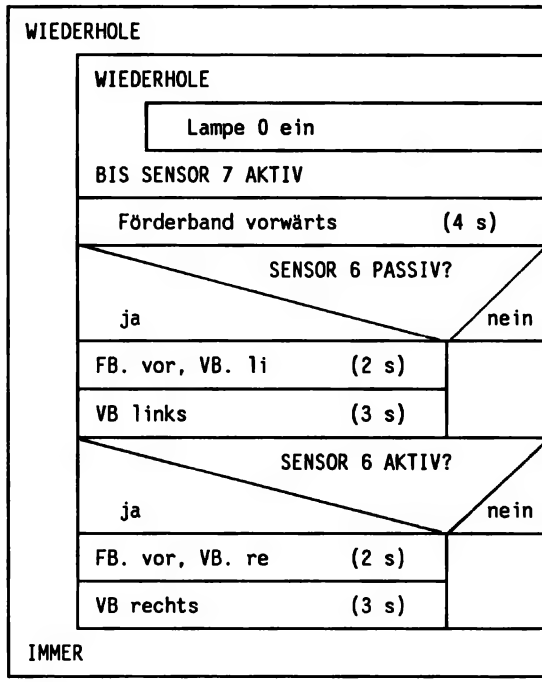


Abb. 4.55 Struktogramm SORT1

Die Zeitspanne, die nötig ist, um einen am Anfang des Förderbands aufgelegten Radiergummi bis zum Sensor zu transportieren, wird von den Schülern experimentell ermittelt.

Die Zeitangabe von 0,1 Sekunden in der ersten Programmzeile "Lampe ein" ist nur im Ablaufmodus auf IBM-PC's und Kompatiblen notwendig, im Sichtlaufmodus und bei Commodore-Computern kann sie ganz entfallen. Vor allem auf Computern mit hoher Taktfrequenz kann es sonst passieren, daß die Strukturanweisung WIEDERHOLE...BIS den Sensor 7 abfragt, bevor die an Kanal 0 angeschlossene Lampe ihre volle Leuchtkraft erreicht hat. Der Sensor meldet dann fälschlicherweise einen aktiven Signalpegel und das Förderband läuft an, noch ehe der erste Radiergummi aufgelegt wurde.

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
Lampe ein	*	*	0	0	0	0	0	1	0,1
WIEDERHOLE									
BIS	1	*							
FB vor	*	*	0	0	1	0	1	0	4,0
WENN	*	0							
FB vor VB li	*	*	1	0	1	0	1	0	2,0
VB links	*	*	1	0	0	0	1	0	3,0
ENDE WENN									
WENN	*	1							
FB vor VB re	*	*	0	1	1	0	1	0	2,0
VB rechts	*	*	0	1	0	0	1	0	3,0
ENDE WENN									
IMMER									

Abb. 4.56 Programmraster SORT1

Eigentlich scheint es auf den ersten Blick völlig unerheblich zu sein, ob mit den Strukturanweisungen WENN...ENDE WENN zuerst auf hohes oder niedriges Sortiergut abgefragt wird. Wenn Sie jedoch Ihr Programm so umstellen, daß entgegen dem obigen Beispiel zuerst auf hohes Sortiergut abgeprüft wird, so werden Sie feststellen, daß nach erfolgtem Rechtstransport eines hohen Radiergummis das Verteilerband auch noch eine Runde leer nach links läuft. Warum wohl?

In den Programmvorschlägen des LEGO Lehrerhandbuchs wird ganz auf den Startsensor verzichtet. Die Sortieranlage läuft, sozusagen im Leerlauf, immer nach links, erst ein hoher Radiergummi schaltet das Verteilerband kurzzeitig auf Rechtslauf.

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
La u Mot ein	*	*	1	0	1	0	1	0	
WIEDERHOLE									
BIS	*	1							
FB vor VB re	*	*	0	1	1	0	0	0	2,0
VB rechts	*	*	0	1	0	0	0	0	4,0
IMMER									

Abb. 4.57 Programmraster SORT2

Der freigewordene Sensor kann jetzt am rechten Ende des Verteilerbandes das Passieren des hohen Sortierguts kontrollieren. Der Rechtslauf des Verteilerbandes erfolgt damit nicht mehr zeitgesteuert.

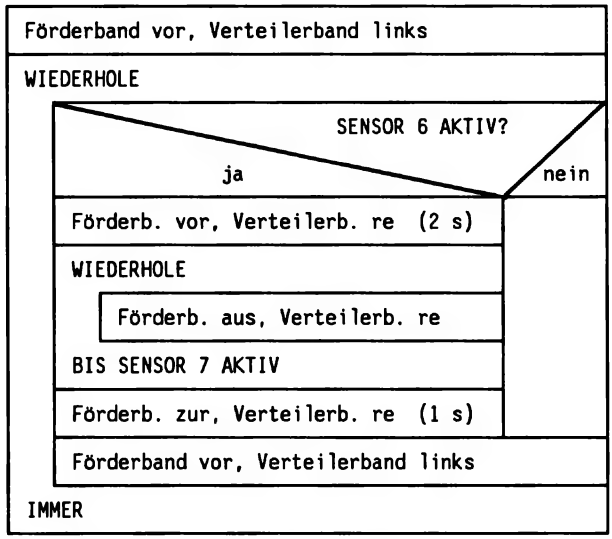


Abb. 4.58 Struktogramm SORT3

Kleine Unregelmäßigkeiten beim Übergang vom Verteiler- zum Förderband werden dadurch ausgeglichen, daß das Zurückschalten des Verteilerbandes auf Linkslauf nun erst erfolgen kann, nachdem der hohe Radier-

gummi tatsächlich das Verteilerband am rechten Ende verlassen hat. Der endgültige Programmvorschlag aus dem Lehrerhandbuch sieht zudem eine kurze Rücktransportphase des Förderbandes vor, damit auch dicht aufeinanderfolgendes unterschiedliches Transportgut richtig sortiert wird.

	7	6	5	4	3	2	1	0	
FB vor VB li	*	*	1	0	1	0	1	0	0,1
WIEDERHOLE									
WENN	*	1							
FB vor VB re	*	*	0	1	1	0	0	1	2,0
WIEDERHOLE									
FB aus VB re	*	*	0	1	0	0	0	1	
BIS	1	*							
FB zur VB re	*	*	0	1	0	1	0	0	1,0
ENDE WENN									
FB vor VB li	*	*	1	0	1	0	1	0	0,1
IMMER									

Abb. 4.59 Programmraster SORT3

4.5.4 Die komplexe Sortieranlage

4.5.4.1 Vorüberlegungen zur Sortierfolge

Bisher war mit der aufgebauten Sortieranlage ein Sortieren nur nach einem einzigen Kriterium möglich. Dabei wurde nach der Höhe des ankommenden Gutes unterschieden, hohes Transportgut wurde von niedrigem getrennt. Im folgenden soll die Aufgabenstellung erweitert und eine Sortierung nach **zwei** Kriterien möglich werden. Die Suche nach anderen Merkmalen liefert eine Vielzahl von Eigenschaften, die ebenfalls als Unterscheidungskriterien dienen könnten. Schüler nennen neben der Größe die Farbe, das Gewicht oder das Material als weitere grundsätzlich mögliche Kriterien. Hier wird zunächst eine Besinnung auf die durch das LEGO Material vorgegebenen Möglichkeiten notwendig: Durch die

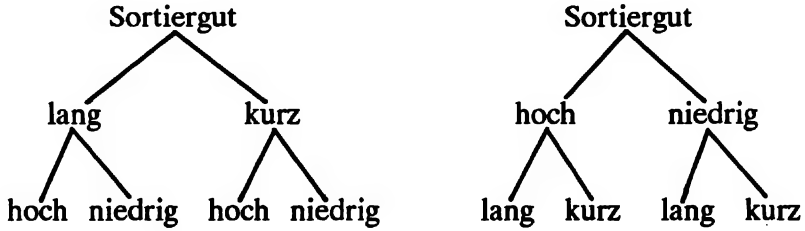
Art der zur Verfügung stehenden Sensoren kommen nur geometrische Eigenschaften für das Sortieren in Betracht. Neben der Höhe bietet sich auch die Länge als Unterscheidungsmerkmal an. Die Problemstellung kann dann etwa folgendermaßen formuliert werden:

Aufgabe: Das Transportgut soll nach den Kriterien **Höhe** und **Länge** sortiert werden.

Hieraus folgt sofort, daß insgesamt vier Merkmale erkannt und unterschieden werden müssen:

	lang	kurz
hoch		
niedrig		

Da eine LEGO Sortieranlage maximal ein Eigenschaftspaar bearbeiten kann, müssen zur Lösung des Problems sicher mehrere Anlagen zusammenarbeiten. Eine erste Sortieranlage kann das Sortiergut beispielsweise nach dem Merkmalspaar lang-kurz sortieren. Man benötigt dann noch zwei weitere Anlagen, die das vorsortierte Gut weiter nach den Merkmalspaaren hoch und niedrig unterscheiden. Natürlich ist auch zuerst eine Aufteilung nach der Höhe und erst dann nach der Länge möglich. Beide Vorgehensweisen sind hier graphisch in Baumdiagrammen dargestellt:



Da von jedem Knotenpunkt aus genau zwei Wege weiterführen, nennt man diese Baumdiagramme auch Binärbäume. Jede Verästelung entspricht dabei einer Sortieranlage. Es ist somit unschwer zu erkennen, daß insgesamt drei Anlagen benötigt werden.

Für die praktische Lösung wird man sich für eine der gezeigten Alternativen entscheiden. Die Unterscheidung nach der Höhe wurde bereits erfolgreich durchgeführt, sie ist für uns "Stand der Technik". Die Sortierung nach der Länge ist dagegen neu und durch die Tatsache, daß dabei hohes wie niedriges Gut gleichermaßen erfaßt werden muß, sicherlich auch problematischer. Es ist deshalb sinnvoll, sich für die erste Alternative zu entscheiden, da bei ihr die kritische Unterscheidung nach der Länge nur einmal erfolgen muß.

Als Sortiergut eignen sich am besten Radiergummiabschnitte, die nach den folgenden Maßangaben aus größeren Läufer-Plast-Radiergummis ausgeschnitten werden:

Querschnitt:	hoch	10 mm × 10 mm
	flach	5 mm × 10 mm
Länge:	lang	31 mm
	kurz	24 mm

Die Möglichkeit, entspechend dimensioniertes Sortiergut schnell und einfach herstellen zu können, spricht ebenfalls für die Verwendung von Radiergummis.

4.5.4.2 Längenunterscheidung mit zwei Sensoren

Es müssen jetzt, unabhängig von ihrer Höhe, kurze und lange Radiergummiabschnitte sicher unterschieden werden können. Zur Lösung dieses Problems sollten zunächst Schülervorschläge gesammelt werden. Teilt man zuvor die entsprechend zugeschnittenen Radiergummiabschnitte aus, so erhalten die Schüler damit die Gelegenheit, die Richtigkeit ihrer Überlegungen sofort nachprüfen zu können; zudem wird die weitere Hypothesenbildung unterstützt. Sobald die Schüler auf die Idee kommen, zwei Optosensoren direkt hintereinander aufzubauen, werden sie erkennen, daß ein kurzer Radiergummi immer nur eine Sensoröffnung abdeckt, während ein langer Abschnitt beim Vorbeitransport kurzzeitig beide Öffnungen verdecken kann.

In einem ersten Lösungsgang werden somit beide Sensoren zur Lang-Kurz-Unterscheidung eingesetzt. Zwei Probleme sind hierbei zunächst noch offen.

Dadurch, daß beide Optosensoren mit den zugehörigen Leuchsteinen in unmittelbarer Nachbarschaft als Lichtschranken verwendet werden, muß Streulicht von einer Lichtschranke zur anderen ausgeschlossen werden. Dies kann recht einfach dadurch erreicht werden, indem ein kurzer LEGO Stein mit zwei Knöpfen quer vor jeden Leuchtstein gesetzt wird. Die Bohrung im LEGO Stein bündelt dann das Licht der Lampen ausreichend.

Weiterhin muß gewährleistet sein, daß sowohl hohes wie auch flaches Sortiergut auf dem Förderband von den Lichtstrahlen erfaßt wird. Deshalb müssen die Optosensoren und die Leuchtsteine tiefer als bisher angebracht werden.

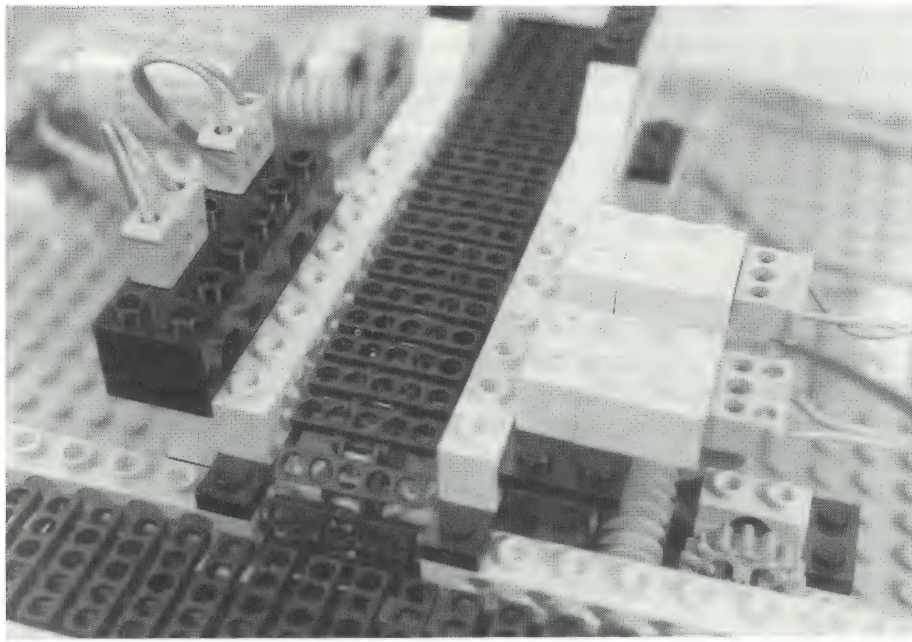


Abb. 4.60 Anbringung der Sensoren

Die Reihenfolge der Sensoren wird so festgelegt, daß auf dem Transportband bewegtes Gut zuerst an Sensor 7 und dann an Sensor 6 vorbeitransportiert wird.

	7	6	5	4	3	2	1	0	
FB vor VB li	*	*	1	0	1	0	0	1	0,1
WIEDERHOLE									
WENN	*	1							
FB vor VB re	*	*	0	1	1	0	0	0	2,0
FB aus VB re	*	*	0	1	0	0	0	0	4,0
FB zur VB re	*	*	0	1	0	1	0	0	1,0
ENDE WENN									
FB vor VB li	*	*	1	0	1	0	0	1	
IMMER									

Abb. 4.61 Programmraster LENSORT (Vorversion)

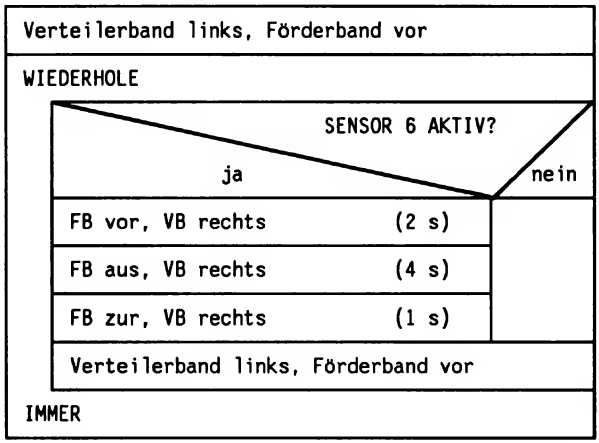


Abb. 4.62 Struktogramm LENSORT (Vorversion)

Hier bedeutet "SENSOR 6 AKTIV", daß hohes Sortiergut die Lichtschranke passiert, das Verteilerband wird umgeschaltet. Ein Testlauf

dieses Programmes demonstriert dessen grundsätzliche Funktionsfähigkeit.

Da beide Sensoren zur Erkennung benötigt werden, muß das vorhandene Programm SORT3 so abgeändert werden, daß der Sensor am Ende des Verteilerbandes entfallen kann. Wir entfernen daher die Strukturanweisung WIEDERHOLE...BIS und versehen die zuvor damit eingeklammerte Zeile: "FB aus VB re" im Wertefeld mit einer Zeitdauer von 4 Sekunden.

Dieses Programm muß jetzt für die neue Aufgabenstellung abgeändert werden. Nun kann "SENSOR 6 AKTIV" lediglich melden, daß Sortiergut, egal ob kurz oder lang, an der zweiten Lichtschranke angekommen ist. In genau diesem Moment muß auch die erste Lichtschranke, also Sensor 7, ebenfalls abgefragt werden: Sind beide aktiv, so handelt es sich um einen langen Radiergummiabschnitt und das Verteilerband muß umgeschaltet werden. Ist dies nicht der Fall, so ist es ein kurzer Abschnitt und das Verteilerband kann in der ursprünglichen Richtung weiterlaufen. Aus diesen Überlegungen wird das folgende Struktogramm abgeleitet.

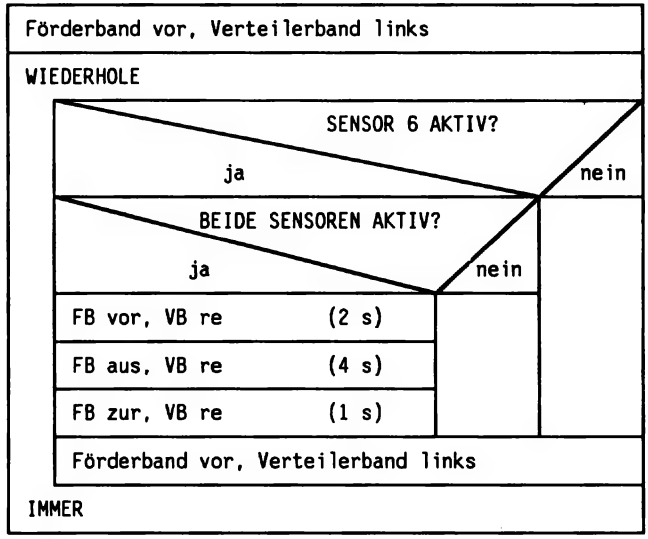


Abb. 4.63 Struktogramm LENSORT1

Leider wird das daraus erstellte LEGO Lines Programm auf einem Commodore-Computer nicht fehlerfrei arbeiten. Die Arbeitsgeschwindigkeit dieser Computergeneration ist einfach zu niedrig, um eine so schnell aufeinanderfolgende Abfrage der Sensoren zu ermöglichen, wie sie hier notwendig ist. Daran ist nicht zuletzt auch die Tatsache mitschuldig, daß die Programme auf diesen Computern nur im langsamen Sichtmodus ablaufen können. Die Bildschirmausgabe benötigt dabei soviel Rechenzeit, daß selbst schnelle IBM-PC's oder Kompatible der AT-Klasse mit einer Taktfrequenz von 12 MHz im Sichtmodus keine fehlerfreie Sortierung garantieren können. Deshalb muß für diese zeitkritische Aufgabe der Längenunterscheidung das Programm im Ablaufmodus gestartet werden.

	7	6	5	4	3	2	1	0	
FB vor VB li	*	*	1	0	1	0	1	1	0,1
WIEDERHOLE									
WENN	*	1							
WENN	1	1							
FB vor VB re	*	*	0	1	1	0	0	0	2,0
FB aus VB re	*	*	0	1	0	0	0	0	4,0
FB zur VB re	*	*	0	1	0	1	0	0	1,0
ENDE WENN									
ENDE WENN									
FB vor VB li	*	*	1	0	1	0	1	1	0,1
IMMER									

Abb. 4.64 Programmraster LENSORT1

4.5.4.3 Längenunterscheidung mit einem Sensor

Als problematisch kann nun die zeitgesteuerte Umschaltung des Verteilerbandes angesehen werden. Sobald sich langes Transportgut auf dem Band verklemmt, wird dieses nicht ordnungsgemäß sortiert, da das Verteilerband nach einer festgelegten Zeitspanne wieder in die Ursprungsrichtung nach links zurückschaltet. Hier wäre es notwendig, mit dem

zweiten Sensor das Passieren des langen Radiergummis am Ende des Verteilerbands abzufragen, wie es bereits beim Sortieren nach der Höhe geschehen ist.

Dazu plazieren wir den Sensor 6 mitsamt seinem am Eingabekanal 1 angeschlossenen Leuchtstein ans rechte Ende des Verteilerbandes und erhalten somit eine Lichtschranke, die das Herabfallen eines langen Radiergummis vom Verteilerband melden kann.

Dies bedeutet aber, daß eine Unterscheidung nach der Länge des Sortierguts auch mit einem Sensor möglich sein muß. Die Frage ist nun, wie mit nur einem Sensor zwischen langen und kurzen Radiergummiabschnitten unterschieden werden kann. Eine Lösungsidee kann dahingehend formuliert werden, daß ein langer Abschnitt für seinen Vorbeitransport am Sensor eine längere Zeitspanne benötigt als ein kurzer Abschnitt.

Sensor 7 meldet auf dem Förderband ankommendes Sortiergut. Dieses wird eine definierte Zeitspanne weitertransportiert. Ist danach Sensor 7 noch aktiv, so handelt es sich um einen langen Radiergummiabschnitt, ist er nicht mehr aktiv, war es ein kurzer Abschnitt.

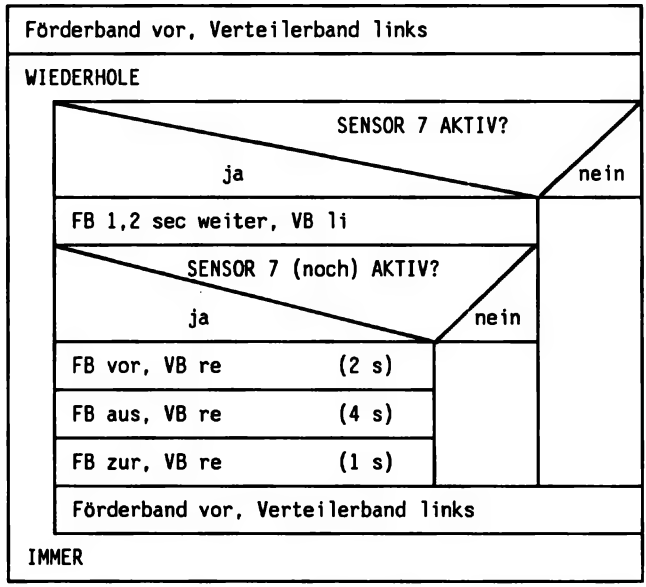


Abb. 4.65 Stuktogramm LENSORT2 (vorläufige Version)

Die Länge der zur sicheren Unterscheidung nötigen Zeitspanne kann von den Schülern wieder durch Probieren herausgefunden werden. Diese Zeitspanne ist natürlich stark von der Geschwindigkeit des Förderbands abhängig, sie kann deshalb von Modell zu Modell variieren. Beim An- und Abbauen der Lichtschranken am Förderband sowie beim Auflegen des Transportguts muß darauf geachtet werden, daß auf den Anfang des Förderbands keine große Belastung ausgeübt wird. Die Getriebeachsen können sonst in ihren Lagern leicht verkanten und durch die dadurch entstehende Reibung die Bandgeschwindigkeit stark vermindern. Das Modell muß dann einfach neu ausgerichtet werden; es ist auch empfehlenswert, den Anfang des Förderbands auf der dem Motorblock gegenüberliegenden Seite durch ein zusätzliches Standbein abzustützen.

	7	6	5	4	3	2	1	0	
FB vor VB 1i	*	*	1	0	1	0	0	1	
WIEDERHOLE									
WENN	1	*							
FB vor VB 1i	*	*	1	0	1	0	0	1	1,2
WENN	1	*							
FB vor VB re	*	*	0	1	1	0	0	0	2,0
FB aus VB re	*	*	0	1	0	0	0	0	4,0
FB zur VB re	*	*	0	1	0	1	0	0	1,0
ENDE WENN									
ENDE WENN									
FB vor VB 1i	*	*	1	0	1	0	0	1	
IMMER									

Abb. 4.66 Programmraster LENSORT2 (vorläufige Version)

Da mit diesem Programm nur ein Sensor für die Unterscheidung von kurzen und langen Bausteinen nötig ist, kann dieser wie im Programm SORT3 zum Abschalten des Verteilerbandes verwendet werden. Hierzu muß im obigen Programm lediglich die Zeile "FB aus VB re " mit der

bereits in SORT3 verwendeten Strukturanweisung WIEDERHOLE...BIS eingeklammert werden. Die Angabe der Zeitdauer entfällt wieder.

	7	6	5	4	3	2	1	0	
FB vor VB li	*	*	1	0	1	0	0	1	0,1
WIEDERHOLE									
WENN	1	*							
FB vor VB li	*	*	1	0	1	0	0	1	1,2
WENN	1	*							
FB vor VB re	*	*	0	1	1	0	1	0	2,0
WIEDERHOLE									
FB aus VB re	*	*	0	1	0	0	1	0	
BIS	*	1							
FB zur VB re	*	*	0	1	0	1	0	0	1,0
ENDE WENN									
ENDE WENN									
FB vor VB li	*	*	1	0	1	0	0	1	
IMMER									

Abb. 4.67 Programmraster LENSORT2

4.5.4.4 Aufbau der kompletten Anlage

Die Aufstellung der Sortieranlagen muß so erfolgen, daß das vom Verteilerband der ersten Anlage herabfallende Gut in die Schüttöffnungen der anderen Bänder fällt. Dafür muß die erste Anlage etwas erhöht postiert werden. Hierzu können die Standbeine der Sortieranlage etwas verlängert oder die Anlage auf Holzbrettchen gestellt werden.

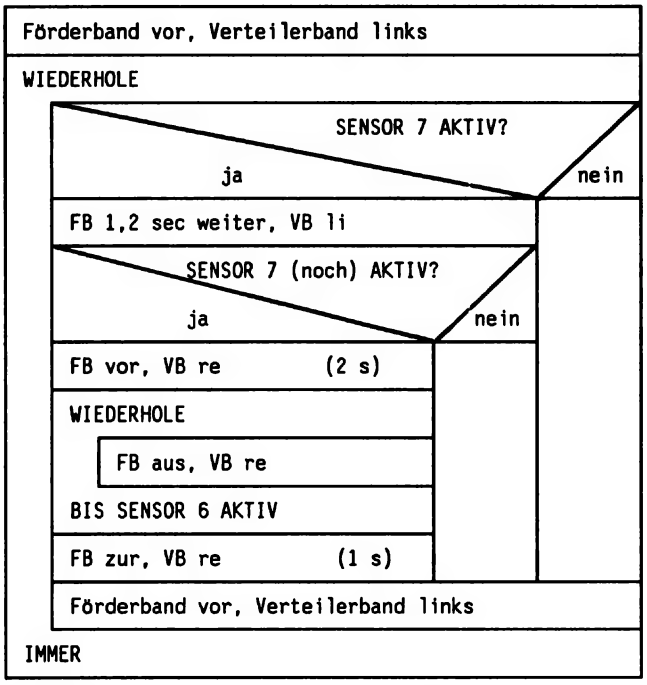


Abb. 4.68 Struktogramm LENSORT2

4.6 Rechnerkopplung

Im vorigen Beispiel wurde eine Möglichkeit des Zusammenspiels mehrerer Steuerungsanlagen demonstriert. Allerdings arbeiten dort alle Anlagen völlig unabhängig voneinander, eine Kopplung untereinander war nicht vorgesehen und für die gewählte Aufgabenstellung auch nicht nötig. In der Realität ist es jedoch gerade diese Kopplung mehrerer Rechnersysteme, welche die hohe Effizienz des Computereinsatzes ausmacht. Zielvorstellung beim Computer Integrated Manufacturing (CIM) ist es gar, alle Verwaltungs- und Herstellungsschritte bei industriellen Fertigungsprozessen computergesteuert ablaufen zu lassen. Nachdem die einzelnen sogenannten C-Techniken, wie CAD, CAM, CAE usw., bereits einen recht hohem Standard erreicht haben, ist es zum Erreichen des genannten Ziels notwendig, diese Einzelpakete für einen Datenaustausch untereinander zu vernetzen. Dies kann selbstverständlich nur über die Kopplung aller beteiligter Computer und Anlagenrechner erfolgen. Die Möglichkeit, mit LEGO Lines Computersteuerungen untereinander zu verbinden, soll deshalb im folgenden kurz dargestellt werden.

Jede Datenübertragung funktioniert grundsätzlich nach dem Schema: Sender-Kanal-Empfänger. Für einen funktionierenden Datenaustausch müssen alle drei Elemente natürlich aufeinander abgestimmt sein. Für eine Übertragung auf optischem Wege verfügen wir mit dem Optosensor über einen geeigneten Empfänger, die Leuchtsteine sind die dazu passenden Sender. Eine Kopplung zweier oder mehrerer Computer kann also ganz einfach dadurch erfolgen, daß ein Leuchtstein des einen Computers mit einem Optosensor des anderen zusammengebaut wird. Die damit verbundenen Möglichkeiten wollen wir hier nur kurz an einem kleinen Beispiel aufzeigen und weitere mögliche Anwendungen Ihrer eigenen Phantasie überlassen.

Für die Aufgabenstellung greifen wir auf die bereits bekannte Sortieranlage zurück. Diese soll jetzt allerdings durch den LEGO Roboter (Bauanleitung 1090 E) mit Sortiergut beschickt werden. Hierfür ist eine Verbindung zwischen beiden Steuerungscomputern durchaus sinnvoll: Die Verteileranlage soll dem Roboter signalisieren, wann sie zur Aufnahme neuen Sortiergutes bereit ist und der Roboter soll nach dem Laden mit einer Fertig-Meldung den Sortiervorgang starten.

Beide Modelle werden nach Anleitung zusammengebaut und so aufgestellt, daß sich der Roboterarm in seiner Endstellung über der Schüttöffnung des Förderbandes befindet. Die Position, in der das Sortiergut aufgenommen werden soll, befindet sich 90° im Uhrzeigersinn von der Schüttöffnung entfernt; dort bauen wir eine kleine Rampe, von welcher der Roboterarm die Radiergummiabschnitte aufnehmen kann. Beim Programmstart befindet sich der Roboter mit geöffnetem Greifarm genau über dieser Rampe. In dieser Ausgangsstellung muß so lange gewartet werden, bis das Startsignal vom Förderband kommt. Dann schließt der Greifer, bringt den Radiergummi über die Schüttöffnung und läßt ihn auf das Förderband fallen. Jetzt kann die Fertig-Meldung vom Roboter an die Sortieranlage erfolgen, der Roboterarm wieder in die Ausgangsstellung zurückgeschwenkt und dort auf ein neues Startsignal gewartet werden.

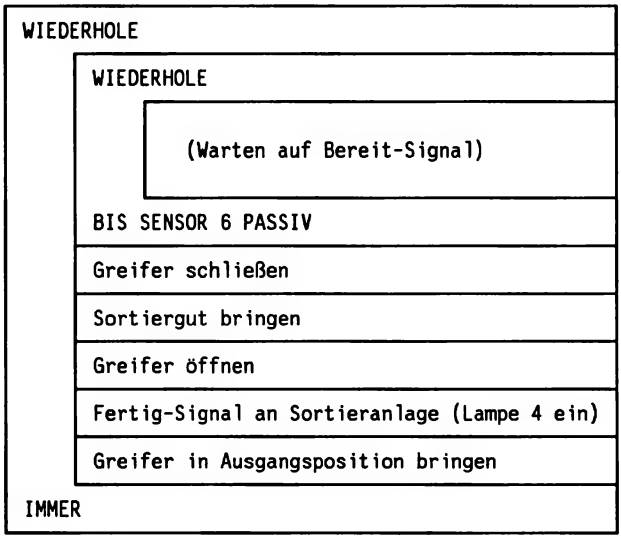


Abb. 4.69 Struktogramm ROBOTER

Da die Bauanleitung des Robotermodells den Eingabekanal 7 für die Schrittsteuerung des Greifarms verwendet, schließen wir den Optosensor als Empfänger für das Bereitsignal **vom** Förderband an Eingang 6 an. Der Leuchtstein für die Fertigmeldung **an** die Sortieranlage wird mit Ausgabekanal 4 verbunden.

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
WIEDERHOLE									
BIS	*	0							
schließen	*	*	0	0	0	0	1	0	0,2
bringen	*	*	0	0	1	0	0	0	
ZÄHLE	1	*							24
öffnen	*	*	0	0	0	0	0	1	0,2
fertig	*	*	0	1	0	0	0	0	0,5
Ausgangspos	*	*	0	0	0	1	0	0	
ZÄHLE	1	*							24
IMMER									

Abb. 4.70 Programmraster ROBOTER

Für den Aufbau und die Programmierung der Sortieranlage können wir auf das Beispiel SORT1 aus dem Abschnitt über das programmgesteuerte Sortieren zurückgreifen. Dort hatten wir eine Anlage realisiert, welche beim Einbringen des Sortiergutes über die Aktivierung des Eingabekanals 7 gestartet wurde. Wenn wir jetzt diesen Sensor 7 mit der Bereit-Lampe (Ausgabekanal 4) des Roboters und den zur Lichtschranke gehörenden Lampenbaustein (Ausgabekanal 1) mit dem Empfangssensor (Eingabekanal 6) des Roboters verbinden, haben wir beide Anlagen bereits hardwaremäßig gekoppelt. Das Zusammenbauen von Lampe und Sensor muß dabei so erfolgen, daß der Lampenbaustein genau vor die ovale Öffnung des Sensors gesetzt wird. Solange die Lampe nicht leuchtet, wird der Sensoreingang aktiviert sein, das Einschalten der Lampe schaltet den Eingang passiv. Da das Fertig-Signal des Roboters durch das Aufleuchten der Lampe dargestellt wird, muß deshalb im Programmraster SORT1 die Abfrage von Sensor 7 auf seinen **passiven** Zustand hin abgeändert werden. Die erste Anweisung: "Lampe ein" ist nun die Bereit-Meldung an den Roboter, der Kommentar wird entsprechend abgeändert und die Zeitspanne aus dem Wertefeld entfernt.

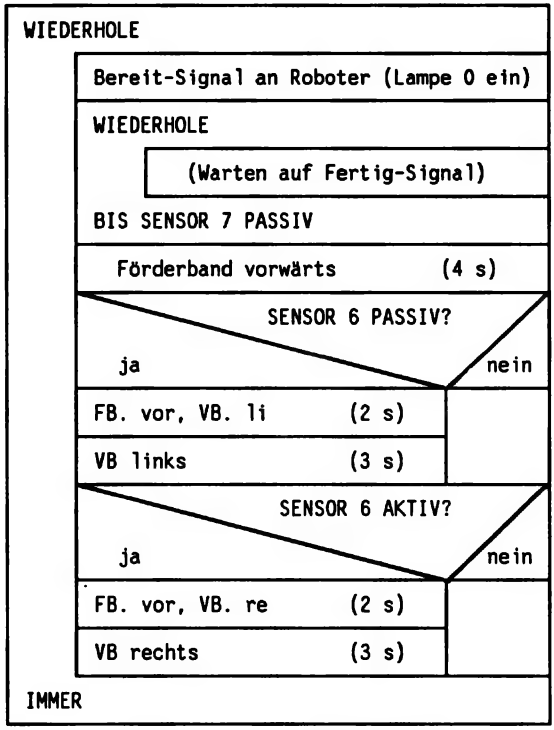


Abb. 4.71 Struktogramm KOPPLUNG

Es bleibt Ihrer Kreativität überlassen, die hier aufgezeigten Möglichkeiten der Rechnerkopplung in eigenen Projekten weiter auszubauen. Von der Firma LEGO gibt es in einem NEWSLETTER beispielsweise einen Vorschlag über eine Parallelvernetzung dreier Computer. Darin werden zwei Computer als Anlagenrechner für je eine Straßenkreuzung und der dritte Computer als Leitrechner eingesetzt. Insgesamt eröffnen sich mit der Möglichkeit der Rechnerkopplung vielfältige Möglichkeiten, die unter LEGO Lines sehr einfach genutzt werden können. Wie einfach diese Verbindung auch zwischen verschiedenen Computertypen hier funktioniert, wird vor allem derjenige ermessen können, der bereits auf herkömmlichem Wege versucht hat, Daten zwischen einem Commodore und einem IBM-PC auszutauschen.

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
bereit	*	*	0	0	0	0	0	1	
WIEDERHOLE									
BIS	0	*							
FB vor	*	*	0	0	1	0	1	0	4,0
WENN	*	0							
FB vor VB li	*	*	1	0	1	0	1	0	2,0
VB links	*	*	1	0	0	0	1	0	3,0
ENDE WENN									
WENN	*	1							
FB vor VB re	*	*	0	1	1	0	1	0	2,0
VB rechts	*	*	0	1	0	0	1	0	3,0
ENDE WENN									
IMMER									

Abb. 4.72 Programmraster KOPPLUNG

5 Der LEGO Buggy

5.1 Vorüberlegungen und Zusammenbau

Sensoren ermöglichen Steuerungssystemen die Erfassung ihrer Umwelt, sie sind gewissermaßen deren Sinnesorgane. Das Sinnesorgan, das wir Menschen am häufigsten zur Erfassung von Informationen einsetzen, ist unser Auge. Dem hohen Komplexitätsgrad optischer Informationen entsprechend, ist dieses Auge ein höchst kompliziertes Organ und um die vom Auge aufgenommene Information in eben dieser Vielschichtigkeit weiterverarbeiten zu können, ist immer noch ein Gehirn nötig. Die heutigen Möglichkeiten elektronischer Bilderfassung und Gestalterkennung sind im Hinblick auf die Leistungen unseres Gehirns allenfalls rudimentärer Art. Trotzdem, oder vielleicht gerade deshalb, faszinieren uns selbst einfache Systeme zur optischen Meßwertaufnahme. Darin mag auch der Reiz begründet sein, den die Optosensoren des LEGO Technic Control Systems unzweifelhaft ausüben. Zudem erweitert die berührungsfreie Meßwertaufnahme im Vergleich zu den Tastensensoren die Möglichkeiten des Systems enorm; erst durch den Einsatz von Lichtschranken wurde beispielsweise die Sortieranlage in der vorgestellten Form möglich.

Zwar ermöglicht der Optosensor nur eine simple Unterscheidung zwischen Hell und Dunkel, aber bereits damit lassen sich recht ansprechende Aufgaben eindrucksvoll lösen: Klebt man auf einen Bogen aus weißem Zeichenkarton schwarze Streifen aus Tonpapier und führt den Sensor knapp über die Papieroberfläche, so ist eine einfache Mustererkennung möglich. Über hellem Untergrund ist der Eingabekanal inaktiv, sobald man jedoch die Sensoröffnung über das schwarze Papier bringt, wird er aktiviert. Nach diesem Grundprinzip wird beispielsweise die Europäische Artikelnummer, die als Strichcode auf den allermeisten Verpackungen aufgedruckt ist, von modernen Scannerkassen im Supermarkt gelesen.

Wir wollen hier jedoch die Hell-Dunkel-Unterscheidung zur Verwirklichung eines einfachen Leitsystems ausnutzen. Dazu kleben wir auf einen größeren Bogen Zeichenkarton eine ca. 10 cm breite "Fahrbahn" aus schwarzem Tonpapier. Diese Fahrbahn sollte keine plötzlichen Richtungsänderungen aufweisen, sondern sich in großen Radien über's Papier schlängeln. Dabei sind durchaus geschlossene Kurse, beispielsweise in der

Form einer liegenden Acht möglich. Unsere Aufgabe ist nun, ein Fahrzeug zu bauen und mit der nötigen "Intelligenz" zu versorgen, das genau der aufgeklebten schwarzen Papierbahn zu folgen imstande ist.

Für den Bau des Fahrzeuges gibt es zwei Möglichkeiten. Unter der Artikelnummer 1038 ist der Bausatz für den LEGO Buggy erhältlich. Dieser Buggy ist ein dreirädriges Gefährt, dessen Haupträder jeweils einzeln von einem Motor über ein Untersetzungsgetriebe angetrieben werden. Das dritte Rad dient lediglich als Stütze und wird wie ein Teewagenrad geschleppt. Damit ist dieser Buggy genau das richtige Fahrzeug für unsere Zwecke, läßt er sich doch durch einfaches Ein-, Aus- und Umschalten der Antriebsmotoren über das Interface programmgesteuert lenken. Da die Technic Control Sets ebenfalls alle benötigten Bauteile enthalten, steht auch der eigenen Kreativität nichts im Wege und man kann leicht selbst ein Fahrzeug nach dem angegebenen Funktionsprinzip konstruieren und zusammenbauen.

Bei der Anbringung der Sensoren muß ohnehin noch getüftelt werden. Die beiden Optosensoren müssen nämlich so an die rechte und linke Vorderseite des Gefährts angebracht werden, daß ihre ovale Öffnung maximal 2 Millimeter über dem Untergrund entlanggeführt wird. Dabei sollten sich, wenn das Fahrzeug genau auf der "Straße" steht, beide Sensoren über dem schwarzen Tonpapier befinden und somit beide Eingabekanäle aktiviert sein.

Für die Verbindung mit dem Interface wird folgendes Anschlußschema vereinbart: Der linke Motor wird mit der Anschlußbuchse C, der rechte mit Buchse B verbunden und zwar derart, daß beim Aktivieren der Ausgabekanäle 5 und 3 der Buggy vorwärts fährt. Alleiniges Aktivieren von Kanal 5 läßt dann den Buggy nach rechts drehen; Kanal 3 dreht nach links. Der linke Sensor wird mit dem Eingangskanal 7, der rechte mit dem Eingangskanal 6 verbunden.

Für eine ungehinderte Fahrt sind die vorhandenen Kabel zu kurz. Als Ergänzung sind deshalb längere Kabel (ca. 3 m, Artikelnummer 1347) erhältlich, die zudem farbige Stecker aufweisen, so daß eine Zuordnung zwischen Motor bzw. Sensor und Interfacebuchse leichter möglich ist. Natürlich wird man sich auch mit Litze in Meterware und einem Ummontieren der Stecker leicht selbst weiterhelfen können.

5.2 Das Programm

Mit der vorgeschlagenen Anbringung der Sensoren liegt auch bereits die programmtechnische Lösung der gestellten Aufgabe auf der Hand: Solange sich beide Sensoren über der schwarzen Tonpapierstraße befinden und demnach beide Eingabekanäle aktiviert sind, kann man davon ausgehen, daß der Buggy genau auf der vorgegebenen Route fährt und keine Kurskorrektur notwendig ist.

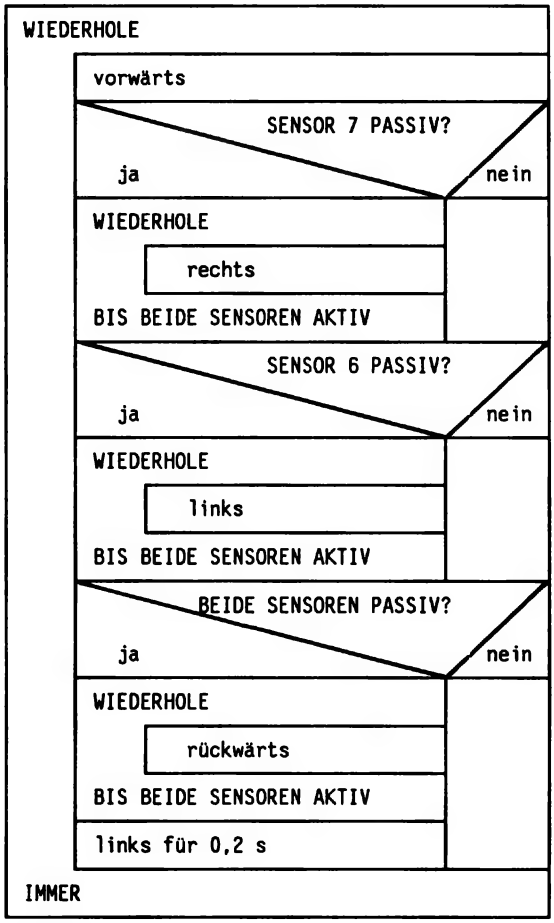


Abb. 5.1 Struktogramm BUGGY

Kommt er jedoch nach rechts von der Straße ab, so erkennt dies der rechte Sensor und der Eingabekanal 6 wird passiv geschaltet. Die logische Konsequenz wird eine Kurskorrektur nach links sein. Entsprechendes gilt für die andere Seite: Sensor 7 meldet ein Abkommen von der vorgegebenen Leitlinie nach links, es muß eine Korrektur nach rechts erfolgen. Beide Kurskorrekturen müssen dabei solange durchgeführt werden, bis beide Sensoren wieder aktiviert sind.

	7	6	5	4	3	2	1	0	
WIEDERHOLE									
vorwärts	*	*	1	0	1	0	0	0	
WENN	0	1							
rechts	*	*	1	0	0	0	0	0	
WIEDERHOLE									
BIS	1	1							
ENDE WENN									
WENN	1	0							
links	*	*	0	0	1	0	0	0	
WIEDERHOLE									
BIS	1	1							
ENDE WENN									
WENN	0	0							
rückwärts	*	*	0	1	0	1	0	0	
WIEDERHOLE									
BIS	1	1							
links	*	*	0	0	1	0	0	0	0,2
ENDE WENN									
IMMER									

Abb. 5.2 Programmraster BUGGY

Da die Zustandsabfrage der Eingabekanäle zeitlich sukzessiv erfolgen muß, ist es durchaus möglich, daß sich plötzlich beide Sensoren über hellem Untergrund befinden, der Buggy also die Leitlinie komplett verlassen hat. In diesem Fall ist es sicherlich das beste, zunächst rückwärts zu fahren, bis die Leitlinie wieder von beiden Sensoren erfaßt wird. Um dann im Vorwärtsgang nicht sofort wieder über's Ziel hinauszuschießen, fügt man sinnvollerweise noch eine kurze Drehung an.

Im Struktogramm (Abb. 5.1) wird der Lösungsansatz nochmals besonders deutlich: Die einzige Anweisung, die bei jeder Wiederholung der Endlosschleife tatsächlich ausgeführt wird, ist der Befehl für die Vorwärtsfahrt. Eine Kurskorrektur wird nur durchgeführt, wenn sie nötig ist. Um dies festzustellen, müssen die Sensoren fortlaufend abgefragt werden. Es leuchtet ein, daß für eine zügige Kurskorrektur diese Abfrage in sehr kurzen Zeitabständen erfolgen muß. Selbst für einen mit 12 MHz getakteten IBM-kompatiblen AT-Computer war die Vorwärtsgeschwindigkeit des Original LEGO Buggies zu groß, um das Programm im Sichtlaufmodus ablaufen lassen zu können. Auf Commodore-Rechnern wird man deshalb nur dann zu befriedigenden Ergebnissen kommen, wenn die Geschwindigkeit des Fahrzeugs durch Hinzufügen einer weiteren Untersetzungsstufe stark vermindert wird.

Leider sind jedoch selbst die mit schnellen Rechnern im Ablaufmodus erreichbaren Ergebnisse in der Regel nicht befriedigend. So kommt es beispielsweise häufig vor, daß Kurskorrekturen nicht rechtzeitig beendet werden, weil der entsprechende Sensor, obwohl wieder über schwarzem Untergrund angelangt, den zugehörigen Eingabekanal nicht aktiviert. Die Ursache hierfür liegt am besonderen Aufbau der LEGO Optosensoren: Durch die interne Beschaltung registrieren diese Sensoren nicht die absoluten Zustände Hell und Dunkel. Sie stellen sich vielmehr beim Anschluß an das Interface selbständig auf die Umgebungshelligkeit ein und reagieren anschließend auf Helligkeitsschwankungen. Bei häufigen Schwankungen, beispielsweise bei vielen schnell aufeinanderfolgenden Kurskorrekturen, regelt die Elektronik auf eine andere Grundhelligkeit ein, was in unserem Fall zu der oben genannten "Fehlfunktion" führen kann. Die gestellte Aufgabe ist jedoch viel zu reizvoll und der Erfolg schon viel zu nah, als daß man jetzt noch die Flinte ins Korn werfen möchte. Zudem ist eine wirklich funktionierende Lösung mit etwas Bastlerfleiß und außerdem sehr preiswert leicht selbst zu realisieren.

5.3 Die Hardware

Das Herzstück des Selbstbausensors ist ein TFK-Reflexkoppler CNY-70, der für ca. 5,-- DM in jedem Elektronik-Laden oder bei Völkner Elektronik (Best.-Nr.:011-712-6) erhältlich ist. Dieser Reflexkoppler enthält in einem kleinen, fast würfelförmigen Gehäuse von rund 7 mm Kantenlänge eine Infrarot-Lumineszenzdiode als Sender sowie einen Silizium-NPN-Fototransistor als Empfänger. Als weitere Bauteile benötigen wir nur noch einen NPN-Transistor BC 548, einen Gleichrichter B80 C800 (od. ähnlich) und drei Widerstände. Die gesamte Schaltung findet auf einer 25×13 mm² kleinen Lochrasterplatine Platz.

An den Eingangsbuchsen 6 und 7 des LEGO Interfaces liegt eine Leerlaufspannung von knapp 9 Volt an. Sobald eine angeschlossene externe Schaltung den Ruhepegel der Spannung verändert, wird dies von der Elektronik im Interface erkannt und als Aktivierung des Eingabekanals ausgewertet. Eine ganz massive Veränderung der Ausgangsspannung erreichen wir beispielsweise mit den Tastensensoren: Jedes Drücken der Taste schließt die Pole der Eingangsbuchse kurz, so daß die Spannung vollständig zusammenbricht. Aber auch kleine Spannungsschwankungen, wie sie von den Optosensoren geliefert werden, reichen bereits für eine Aktivierung eines Eingabekanals aus.

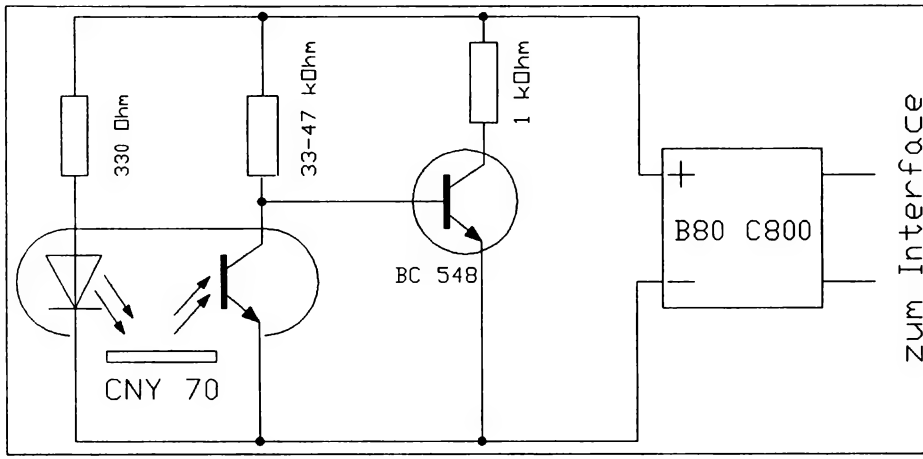


Abb. 5.3 Schaltplan für den Selbstbausensor

Die Lumineszenzdiode des Reflexkopplers wird über einen Schutzwiderstand von 330 Ohm mit Energie versorgt. Wird ihre Strahlung über eine helle Fläche auf die Basis des Fototransistors reflektiert, so steuert dieser durch und zieht die Basis des zweiten NPN-Transistors auf GND; durch diesen Transistor kann somit kein Strom fließen.

Wird keine Strahlung reflektiert, so sperrt der Fototransistor, dafür steuert der externe Transistor durch. Der jetzt durch diesen Transistor und dessen Kollektorwiderstand fließende Strom führt zu einem Spannungsabfall in der Schaltung, der vom Interface als Aktivierung des Eingabekanals erkannt wird.

Um beim Anschluß des Selbstbausensors nicht auf die Polung achten zu müssen, wird noch ein Gleichrichter vorgeschaltet.

Der Aufbau der Schaltung geschieht am einfachsten auf einer kleinen Lochrasterplatine:

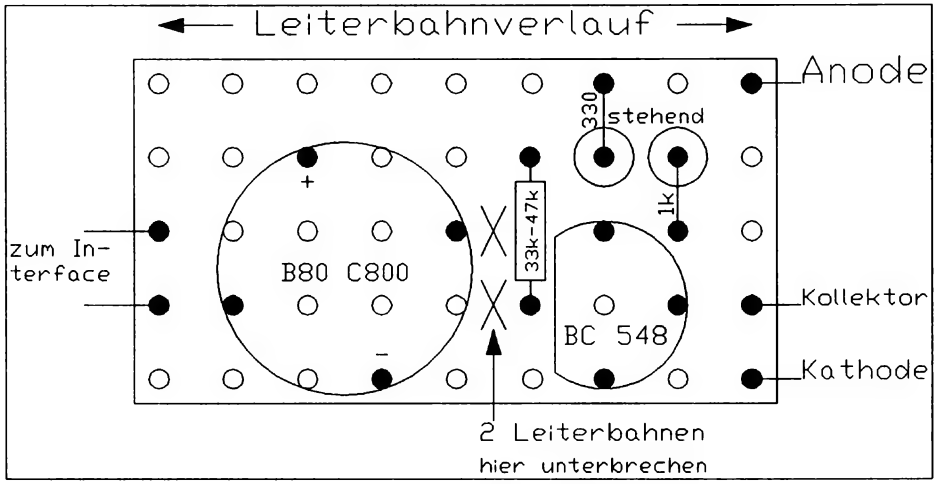


Abb. 5.4 Schaltungsaufbau auf der Lochrasterplatine

An die Anschlußpunkte zum Interface lötet man ein zweiadriges Kabel. An dessen anderes Ende kommen zwei kleine Eisenbahnstecker. Diese Stecker passen genau in die Buchsen des Interfaces oder der LEGO Stecker, eventuell müssen ihre Kontaktstifte vorsichtig aufgefächert

werden. Der Reflexkoppler wird mit einem dreiadrigen Kabel an die Schaltung angeschlossen. Da die Kathode des Senders und der Emitter des Emfängers auf demselben Potential liegen werden, lötet man die entsprechenden Anschlüsse des Reflexkopplers zusammen.

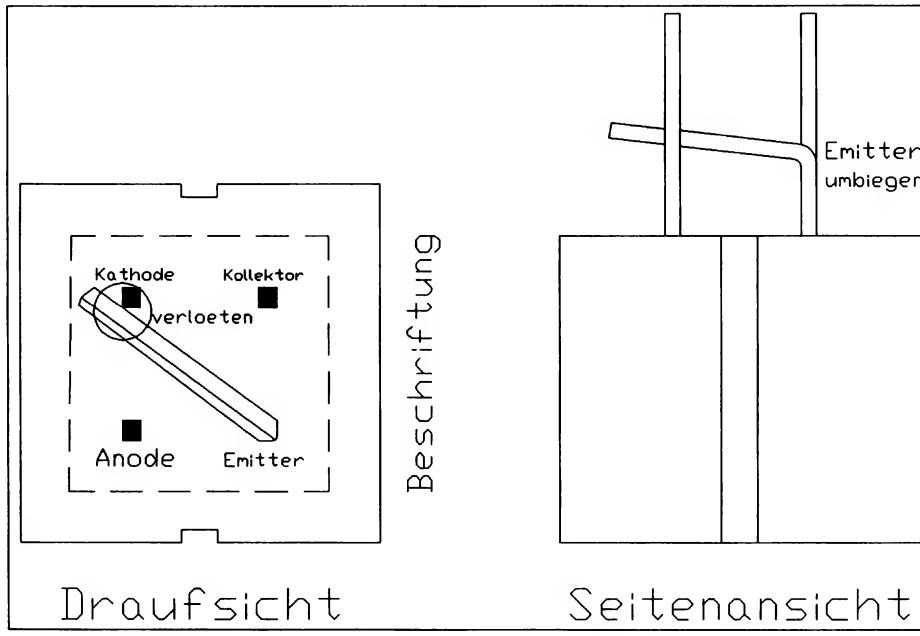


Abb. 5.5 Pinbelegung des Reflexkopplers

Nach der Fertigstellung schließen wir den Selbstbausensor an das Interface an und führen den Reflexkoppler in wenigen Millimetern Abstand über eine schwarz-weiß gefärbte Unterlage. Dabei muß über dunklem Untergrund der Eingabekanal aktiviert werden. Der Spannungspegel der Schaltung sollte sich von ca. 6,5 Volt über reflektierenden Flächen auf etwa 6,2 Volt über dunklem Untergrund verändern. Dies mißt man beispielsweise an den Ausgängen des Gleichrichters. Bei unbefriedigendem Funktionieren kann der Kollektorwiderstand des Fototransistors verändert werden. Je kleiner dessen Wert ist, umso kleiner ist auch der Arbeitsabstand des Kopplers vom reflektierenden Medium. Umgekehrt kann eine größere Entfernung zur reflektierenden Fläche durch Vergrößern dieses Widerstandswerts erreicht werden, wobei jedoch auch die Fremdlichtempfindlichkeit zunimmt. Der angegebene Wertebe-

reich zwischen 33 und 47 kOhm hat sich in praktischen Versuchen bewährt und erlaubt einen Reflektionsabstand von bis zu 5 Millimetern.

Für den Einsatz bei der Spursteuerung befestigt man den Reflexkoppler mit Klebeband an einer Achse des Technic Control Sets. Steckt man diese Achse durch eine Bohrung einer LEGO Platte, so kann der Abstand des Kopplers von der Unterlage stufenlos eingestellt werden. Die Lochrasterplatine mit der Schaltung klebt man auf dem Buggy fest und schließt die Sensoren an die entsprechenden Eingabekanäle an. Das im vorigen Abschnitt erstellte Programm läßt nun den Buggy genau der aufgezeichneten Spur folgen.

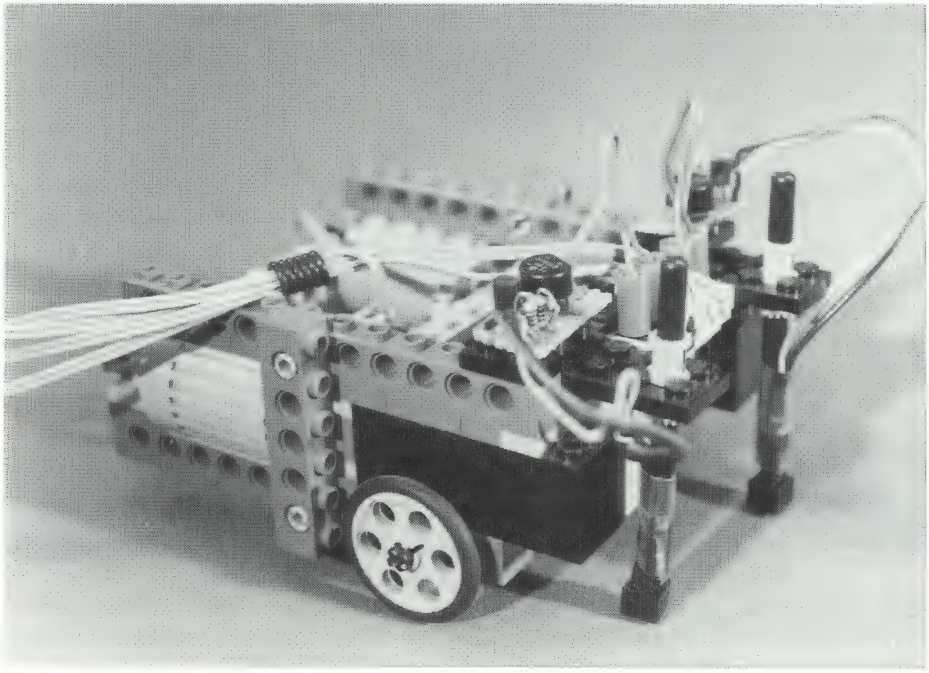


Abb. 5.6 Sensoranbringung am BUGGY

Bauteilliste für den Selbstbausensor:

1 Reflexkoppler CNY 70
1 NPN-Transistor BC 548 (oder ähnlich)
1 Gleichrichter B80 C800 (oder ähnlich)
1 Widerstand 330 Ohm
1 Widerstand 1 kOhm
1 Widerstand 33 kOhm - 47 kOhm
1 Stück Lochrasterplatine
2 Eisenbahnstecker (2mm)
Litze

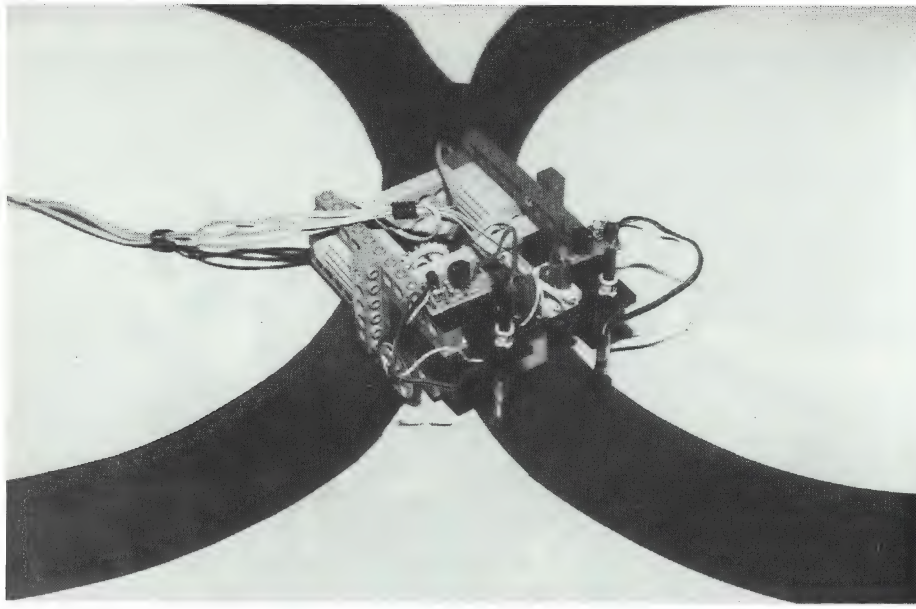


Abb. 5.7 Der Buggy auf der Fahrbahn

6 Programmierung in anderen Programmiersprachen

6.1 Programmierung in BASIC

6.1.1 Hardwareinformationen

Durch seine graphische Benutzeroberfläche sowie die wenigen und aussagekräftigen Strukturanweisungen ist LEGO Lines für den schulischen Einsatz und eine Hinführung an die Thematik des Steuerns mit dem Computer geradezu prädestiniert. Besonders die beiden vorausgegangenen Beispiele, die komplexe Sortieranlage und der LEGO Buggy, stellen zudem die Mächtigkeit des Systems trotz des eingeschränkten Befehlsumfangs eindrucksvoll unter Beweis. Dadurch lassen sich LEGO Lines und das LEGO Technic Control System keinesfalls als Spielerei abqualifizieren, sie bieten vielmehr eine hervorragende und didaktisch durchdachte Möglichkeit der Einführung und Einarbeitung in den Computereinsatz im technisch-physikalischen Bereich.

Die in diesem Band vorgeschlagenen Aufgabenstellungen lassen sich mit den dazugehörigen Programmen im Unterricht der Sekundarstufe I erprobtermaßen realisieren. Hierzu ist das ebenfalls aufgezeigte kleinschrittige Vorgehen notwendig. Die leichte Editierbarkeit von LEGO Lines Programmen ermöglicht eine problemlose Erweiterung erarbeiteter Grundprogramme. Allerdings scheint nach unseren Beobachtungen der Großteil der Schüler bei den vorgestellten komplexen Programmen (AMPEL4, BAHN4, LENSORT2) an der Grenze ihrer Leistungsfähigkeit angelangt zu sein. Hieraus und aus der Funktionalität der erstellten Programme kann sicherlich der Schluß gezogen werden, daß die Möglichkeiten von LEGO Lines für den intendierten schulischen Einsatz mehr als ausreichend sind! Die fehlende Verwendungsmöglichkeit von Variablen dürfte hier kaum einen echten Mangel darstellen.

Hat man sich als Lehrer oder interessierter Schüler jedoch in die Materie eingearbeitet und Spaß an dieser Art des Umgangs mit dem Computer gefunden, so bedeutet der fehlende Variablenzugriff eine Einschränkung in der Arbeit mit LEGO Lines. Um diese Grenze zu sprengen, muß man LEGO Lines verlassen und andere Möglichkeiten der Kommunikation mit dem LEGO Interface suchen. Einige dieser Möglichkeiten sollen nun im folgenden ansatzweise aufgezeigt werden. Gleich an dieser Stelle muß jedoch nochmals ausdrücklich betont werden, daß diese Hinweise

keinesfalls für den schulischen Einsatz gedacht sind, sondern lediglich dem Lehrer oder sonstigen Interessierten zusätzliche Hintergrundinformationen und Anregungen für eine eigene Weiterarbeit geben sollen.

Deshalb können wir auch guten Gewissens trotz der in der Einleitung aufgeführten Vorbehalte zunächst die Programmiersprache BASIC zur Kommunikation mit dem Interface verwenden. Dies nicht zuletzt auch deshalb, weil BASIC auf jedem Computer vorhanden sein dürfte und durch die Implementierung von BASIC als Interpreter erste Versuche einfach und schnell durchgeführt werden können. Für die folgenden Überlegungen wird vorausgesetzt, daß Sie bereits Erfahrungen in der Programmierung mit BASIC gesammelt haben, da an dieser Stelle keine Einführung in diese Programmiersprache erfolgen kann. Es sollen lediglich in kurzen Programmfragmenten einige Möglichkeiten der Modellsteuerung unter BASIC aufgezeigt werden.

Zuvor müssen wir uns jedoch mit den technischen Details vertraut machen (, was bei der Arbeit mit LEGO Lines nicht nötig war!). Am besten rufen Sie sich hierfür den Abschnitt "Steuern mit dem Computer" vom Anfang des Buches nochmals in Erinnerung. Wichtig zu wissen ist, daß sämtliche Information im Computer meistens in achtstelligen Bitmustern aus Einsen und Nullen, den sogenannten Bytes abgelegt sind. Diese Achtstelligkeit finden wir ja auch auf dem Interface mit den sechs Ausgabe- und den zwei Eingabekanälen wieder, woraus wir unschwer schließen können, daß die Kommunikation mit der Außenwelt durch Einlesen oder Ausgeben eines Bytes vom bzw. zum Interface zustandekommt. Dieses Senden und Empfangen der Bytes geschieht über sogenannte Ports des Computers. Alle Ein- und Ausgaben des Rechners (Tastatur, Bildschirm, Drucker, ...) laufen über solche Ports, so daß deren mehrere benötigt werden. Eine gewisse Analogie finden wir in den Briefkästen eines Hochhauses: Jede Familie besitzt einen Briefkasten, der gewissermaßen ihren Eingabeport darstellt. Damit der Briefträger die Briefe richtig zuordnen kann, ist jeder Briefkasten mit dem Namen seines Eigners versehen.

Auch die verschiedenen Ports eines Computers müssen natürlich eindeutig bezeichnet werden, um die ein- und ausgehenden Informationen in die richtigen Bahnen zu lenken. Die Ports werden zu diesem Zweck einfach durchnumeriert, die Nummer eines Ports stellt seine Adresse, die sogenannte Portadresse dar.

Die erste Information, die wir für das Ansprechen des Interfaces außerhalb von LEGO Lines benötigen, ist die Adresse des Ports, an welchen das Interface mit dem Flachbandkabel angeschlossen ist. Genau wie Sie bei jedem Umzug eine neue Adresse erhalten, wird das LEGO Interface in jedem Rechnertyp über eine andere Adresse angesprochen. Im Commodore 64 oder 128 wird das Interfacekabel in den USER-Port eingesteckt und dessen Adresse hat die Nummer 56577. Zusätzlich muß noch die Richtung des Datentransfers im sogenannten Richtungsregister mit der Adresse 56579 angegeben werden. Für die benötigte Aufteilung in zwei Eingabe- und sechs Ausgabekanäle trägt man den Wert 63 in das Richtungsregister ein. Nähere Informationen hierüber entnehmen Sie bitte dem Computerhandbuch.

In IBM-PC's und Kompatiblen wird die LEGO Portkarte eingesteckt, welche die Adresse 925 decodiert. Durch Auftrennen einer Brücke auf der Karte ändert sich diese Portadresse von 925 auf die Adresse 926, so daß bis zu zwei Schnittstellen pro Rechner betrieben werden können.

Durch die unterschiedliche Konzeption beider Computertypen sind auch die entsprechenden BASIC-Befehle für das Ausgeben und Einlesen von Daten über diese Ports verschieden.

Die Ports des Commodore sind "memory mapped". Dies bedeutet, daß die Ports eigentlich Speicherzellen sind, die über elektrische Leitungen eine Verbindung zur Außenwelt erhalten haben. Der Zugriff auf solche Speicherzellen geschieht mit den Befehlen PEEK und POKE.

IBM-Rechner und entsprechende Nachbauten verwenden reale Ports, um Speicherplatz zu sparen. Diese Ports werden mit den Befehlen INP und OUT angesprochen.

Schließlich sollten wir uns nochmals die Binärcodierung der im Computer zu verarbeitenden Informationen ins Gedächtnis zurückrufen. Jedes Bit kann den Zustand EINS oder NULL annehmen. Je nach ihrer Stellung im Byte besitzen die Bits die Stellenwerte 1, 2, 4, 8, 16, 32, 64 und 128 des Dualsystems. Eine Arbeit auf dieser Ebene verlangt somit ein fortwährendes Umrechnen zwischen dem Dual- und dem Dezimalsystem. Ein Beispiel soll diesen Sachverhalt nochmals erläutern:

Aufgabe: Es sollen die Ausgabekanäle 5, 3, 1 und 0 des Interfaces aktiviert werden.

Kanal:	7	6	5	4	3	2	1	0
Muster:	*	*	1	0	1	0	1	1

Zur Lösung schreibt man sich am besten die Stellenwerte der einzelnen Bitpositionen hinzu:

Kanal:	7	6	5	4	3	2	1	0
Wert:	128	64	32	16	8	4	2	1
Muster:	*	*	**1**	**0**	**1**	**0**	**1**	**1**

Berechnung: 1·32 +0·16 +1·8 +0·4 +1·2 +1·1 = 43

Nun muß man die Stellenwerte derjenigen Kanäle, welche aktiviert werden sollen, addieren; der dezimale Ausgabewert 43 schaltet das Interface in den gewünschten Zustand.

Am Commodore löst man die gestellte Aufgabe mit dem Befehl:

POKE 56577,43

Zuvor muß allerdings dem Richtungsregister über

POKE 56579,63

die gewünschte Übertragungsrichtung mitgeteilt werden. Diese Richtungsinitialisierung muß lediglich einmal am Beginn der Arbeit mit dem Commodore ausgeführt werden.

Bei IBM-PC's und Kompatiblen kann die Festlegung der Übertragungsrichtung entfallen. Zur Lösung der gestellten Aufgabe genügt der Befehl:

OUT 925,43

Am besten probieren Sie nun selbst einige Ausgabewerte aus und beobachten die Auswirkungen auf dem Interface. Da nur sechs Ausgabeleitungen vorhanden sind, können nur Zahlenwerte von 0 bis 63 ausgegeben werden, wobei der Wert 63 alle Kanäle einschaltet. Mit der

Ausgabe des Wertes 0 inaktivieren Sie dagegen alle Kanäle. Für unsere weiteren Überlegungen ist es sinnvoll, am Ende dieser Versuche zunächst wieder alle Ausgangskanäle auszuschalten.

6.1.2 Ein- und Ausgabeanweisungen

Wir wollen uns nun anschauen, wie die an den Eingabekanälen des Interfaces anliegende Information abgefragt werden kann. Über die bereits genannten Befehle läßt sich dies recht einfach realisieren:

PRINT PEEK(56577) bzw. PRINT INP(925)

liefert als Ergebnis den Wert 0, wenn keine aktivierten Sensoren an den Eingabebuchsen des Interfaces angeschlossen sind. Für weitere Versuche schließen wir deshalb die beiden Tastensensoren an die Eingabekanäle 6 und 7 an. Eine nochmalige Abfrage nach Aktivieren des Kanals 6 liefert den Wert 64. Fragt man bei aktiviertem Kanal 7 ab, so erhält man als Ergebnis 128. Hält man beide Sensoren gedrückt, so wird 192, die Summe beider Zahlen, angezeigt. Diese Werte sollten kein großes Erstaunen mehr hervorrufen, sind sie doch die Dezimalwerte der beiden höchstwertigen Bitpositionen. Damit könnte in einem BASIC-Programm recht einfach der Zustand der beiden Eingabeleitungen abgefragt werden: Aus dem Eingabewert 64 kann geschlossen werden, daß Kanal 6 aktiviert ist; der Wert 128 meldet entsprechendes für Kanal 7 und beim Wert 192 weiß man, daß beide Sensoren gedrückt sind.

Leider gelten die genannten Zahlenwerte nur, wenn zum Zeitpunkt der Abfrage keine Ausgabekanäle aktiviert sind. Schalten wir mit

POKE 56577,5 bzw. OUT 925,5

die Kanäle 0 und 2 ein und wiederholen nun die Abfrage:

PRINT PEEK(56577) bzw. PRINT INP(925),

so werden je nach dem Zustand der angeschlossenen Sensoren die Zahlenwerte 5, 69, 133 oder 197 geliefert. Dies bedeutet, daß der zuvor an das Interface ausgegebene Zahlenwert beim Lesen der Eingabekanäle ebenfalls miterfaßt wird. Damit entfällt aber die oben zunächst angedeutete Möglichkeit des einfachen Vergleichs mit 0, 64, 128 oder 192. Vielmehr müßte vor einem Vergleich zuerst der zuvor an das Interface ausgegebene Wert vom eingelesenen subtrahiert werden, was wiederum

bedeuten würde, daß jeder Ausgabewert in einem Zwischenspeicher "gemerkt" werden muß. Für eine Abfrage, ob der Eingangskanal 6 aktiv ist, nachdem die Ausgangskanäle 0 und 2 aktiviert wurden, wäre beispielsweise folgendes Vorgehen nötig:

```
130 WERT = 5
140 OUT 925, WERT
150 IF INP(925)-WERT = 64 THEN ...
```

Mit etwas Logik geht es jedoch wesentlich einfacher. Um zu erfahren, ob Kanal 6 aktiv ist, genügt es ja völlig, nur das entsprechende Bit zu untersuchen, alle anderen Bits sind völlig uninteressant. Also wird man die "überflüssigen" Bits ausblenden, was mit der logischen AND-Funktion geschehen kann. Die möglichen Ergebnisse bei der Verknüpfung zweier Bits über die AND-Funktion werden in einer Wahrheitstafel zusammengestellt:

Eingabe		AND
0	0	0
0	1	0
1	0	0
1	1	1

Man erkennt, daß das Ergebnis der AND-Verknüpfung nur dann 1 ist, wenn beide zu verknüpfenden Bits 1 sind. Diese Eigenschaft machen wir uns für das Ausblenden der uninteressanten Bits zunutze. Nach der Ausgabe des Wertes 5 und bei aktiviertem Kanal 6 befindet sich folgendes Bitmuster am Computerport:

Kanal:	7	6	5	4	3	2	1	0
Wert:	128	64	32	16	8	4	2	1
Muster:	0	1	0	0	0	1	0	1

Um alle Bits außer Bit Nummer 6 auszublenden, verknüpft man die Bits des Ports bitweise mit einer entsprechenden Maske über die AND-Funktion.

Bit 6 gesetzt:		Bit 6 nicht gesetzt:	
Portbits:	0 1 0 0 0 1 0 1	Portbits:	0 0 0 0 0 1 0 1
Maske:	0 1 0 0 0 0 0 0	Maske:	0 1 0 0 0 0 0 0
AND	0 1 0 0 0 0 0 0	AND	0 0 0 0 0 0 0 0

Ist am Port das Bit 6 gesetzt, so liefert die AND-Verknüpfung den dezimalen Wert 64, ist das Bit nicht gesetzt, so ist das Ergebnis 0. Als Maske dient einfach diejenige Bitkombination, in welcher nur das Bit gesetzt ist, das überprüft werden soll. In BASIC wird die dargestellte Operation durch die AND-Funktion durchgeführt:

IF PEEK(56577) AND 64 THEN ... bzw.
IF INP(925) AND 64 THEN ...

Der BASIC-Interpreter benötigt für die Auswertung der IF-Verzweigung einen Wahrheitswert. Da der Wahrheitswert FALSCH nur durch den numerischen Wert 0 repräsentiert wird und alle anderen Zahlenwerte als WAHR interpretiert werden, reicht für eine positive Abfrage die obige Kurzschreibweise völlig aus, Konstrukte wie IF (PEEK(56577) AND 64) = 64 THEN ... sind daher bei der positiven Abfrage einer einzelnen Eingabeleitung unnötig. Lediglich beim Vergleich auf den Wert 0 oder bei der Überprüfung, ob beide Sensoren gedrückt sind, muß explizit auf die entsprechenden Werte abgefragt werden, da die AND-Funktion sonst bereits bei einem gesetzten Bit den Wahrheitswert WAHR ergibt. Ein kurzes Beispielprogramm soll die Sensorabfrage nochmals erläutern:

```
10 IF (INP(925)AND192) = 192 THEN PRINT "Beide Sensoren" : GOTO 10
20 IF INP(925) AND 64 THEN PRINT "Sensor 6" : GOTO 10
30 IF INP(925) AND 128 THEN PRINT "Sensor 7" : GOTO 10
40 GOTO 10
```

Programm: SENSOR.BAS

6.1.3 Programmbeispiele

Mit dem erworbenen Wissen können wir nun das Programm für die Fußgängerampel in BASIC formulieren. Obwohl mit Kommentaren versehen und mit Leerzeilen strukturiert, ist dieses Programm sehr schwer lesbar und unterstreicht die in der Einleitung gemachten Bedenken hinsichtlich der Verwendung der Programmiersprache BASIC

nachdrücklich. Die dezimalen Ausgabewerte stehen in keinem erkennbaren Zusammenhang mit den gewünschten Auswirkungen an der Verkehrsampel, die Zeitschleifen müssen zusätzlich programmiert werden. Programmänderungen sind nur in sehr konzentrierter Arbeit sinnvoll vorzunehmen.

```
100 REM FUSSAMP2 als BASIC-Programm
110 :
120 :
130 OUT 925,33 : REM GRÜN rot
140 :
150 REM Sensorabfrage
160 IF INP(925) AND 64 GOTO 190
170 GOTO 160
180 :
190 T=5000 : GOSUB 400
200 :
210 OUT 925,17 : REM GELB rot
220 T=5000 : GOSUB 400
230 :
240 OUT 925,9 : REM ROT rot
250 T=5000 : GOSUB 400
260 :
270 OUT 925,10 : REM ROT grün
280 T=10000 : GOSUB 400
290 :
300 OUT 925,9 : REM ROT rot
310 T=5000 : GOSUB 400
320 :
330 OUT 925,25 : REM ROT-GELB rot
340 T=5000 : GOSUB 400
350 :
360 GOTO 130
370 :
380 :
390 :
400 FOR X = 0 TO T : NEXT X
410 RETURN
```

Programm: FUSSAMP2.BAS

Wie übersichtlich und einfach ist dagegen das entsprechende LEGO Lines Programm FUSSAMP2 (Abb. 4.22) aus Kapitel 4.3.5!

Dabei haben wir in diesem BASIC-Programm nur Dinge realisiert, die auch LEGO Lines bietet; die Möglichkeit der Verwendung von Variablen wurde überhaupt noch nicht angesprochen. Vorbereitend wollen wir zunächst die ZÄHLE-Anweisung in BASIC nachbilden:

```
100 REM Nachbildung der ZÄHLE-Anweisung in BASIC
110 :
120 INPUT "Wieviele Hell-Dunkel-Wechsel"; ZAHL
130 :
140 OUT 925,1 : REM Kanal 0 aktivieren
150 :
160 FOR I = 1 TO ZAHL
170 IF INP(925) AND 64 THEN GOTO 170
180 IF (INP(925) AND 64) = 0 THEN GOTO 180
190 NEXT I
200 :
210 OUT 925,2 : REM Kanal 1 aktivieren
```

Programm: ZAEHLE.BAS

Das Programm fragt nach der Anzahl der zu zählenden Hell-Dunkel-Wechsel und aktiviert dann Ausgabekanal 0. In der FOR-NEXT-Schleife wird die vorgegebene Anzahl der Impulswechsel ausgezählt und danach Kanal 1 aktiviert. Abgefragt wird Sensor 6. Um Eingangsimpulse von Kanal 7 zu zählen, muß lediglich der Wert des Maskenbytes von 64 in 128 geändert werden.

```
100 REM AUFSUMMIEREN DER EINGABEIMPULSE
110 :
120 ZAEHLER = 0 : FLAG = 0 : ALTFLAG = 0 : ENDE = 0
130 :
140 PRINT "Zählen, wie oft Kanal 6 aktiviert wurde"
150 PRINT
160 PRINT "Drehen Sie jetzt die Segmentscheibe an Sensor 6"
170 PRINT "Ende: Sensor 7 aktivieren"
180 :
190 IF INP(925) AND 64 THEN FLAG = 1
200 IF INP(925) AND 128 THEN ENDE = 1
210 IF (INP(925) AND 64) = 0 THEN FLAG = 0
220 IF INP(925) AND 128 THEN ENDE = 1
230 IF FLAG = 1 AND ALTFLAG = 0 THEN ZAEHLER = ZAEHLER + 1
240 ALTFLAG = FLAG
250 IF ENDE = 1 THEN GOTO 280
260 GOTO 190
270 :
280 PRINT "Kanal 6 wurde "; ZAEHLER; " mal aktiviert"
```

Programm: ZAEHLER.BAS

Im obigen Programm wird schließlich eine Möglichkeit aufgezeigt, Sensorimpulse in eine Variable zu zählen. Jedes Aktivieren des an Kanal 6 angeschlossenen Sensors, egal ob Tasten- oder Optosensor inkrementiert nach dem Programmstart die Variable ZAEHLER. Durch ein Aktivieren von Sensor 7 wird die Zählung abgebrochen und das Ergebnis ausgegeben. Damit das Programm jederzeit beendet werden kann, geschieht die

Zählung indirekt über das Setzen eines Flags: Immer wenn ein Wechsel von 0 auf 1 stattgefunden hat, wird die Variable ZAEHLER erhöht.

Die hier aufgezeigten Programmbeispiele sollten Ihnen lediglich als Anregung dienen. Weitere Hinweise, Tips und Programmvorschläge zur direkten Programmierung in BASIC finden Sie im mitgelieferten Handbuch zum Interfacekabel.

6.2 Programmierung in TURBO C

6.2.1 Grundlagen

Mit den am Beginn des vorigen Abschnitts aufgezeigten Grundlagen läßt sich das LEGO Interface über jede Programmiersprache steuern, welche eine direkte Ansprache der Ein-/Ausgabeports erlaubt. Für einen ersten Zugang wurde die Sprache BASIC gewählt, weil sie wohl überall verfügbar und für ein anfängliches Probieren sehr gut geeignet ist. Allerdings kommen bereits in den wenigen Programmbeispielen die bekannten Schwächen von BASIC deutlich zum Ausdruck.

Insbesondere die Sprachen Pascal und C erfreuen sich zur Zeit immer größerer Beliebtheit und Verbreitung. Da zur Sprache Pascal eine spezielle Schnittstelle, der LEGO TC Controller existiert, welcher im zweiten Teil des Buches ausführlich beschrieben wird, soll in diesem Abschnitt auf die Steuerung unter Turbo-C eingegangen werden. Hierbei wird vorausgesetzt, daß Sie bereits Erfahrungen im Umgang mit der Programmiersprache C erworben haben.

Wie schon im vorherigen Abschnitt erläutert, müssen zur Kommunikation mit dem LEGO Interface Daten über die Portadresse 925 ausgetauscht werden. Für diesen Datenaustausch von 8-Bit-Werten über eine Portadresse stellt Turbo-C die beiden folgenden Funktionen zur Verfügung:

unsigned char inportb(int portid)
void outportb(int portid, unsigned char value)

Die Prototypen beider Funktionen befinden sich in der Headerdatei dos.h, die deshalb vom Präprozessor über #include <dos.h> eingebunden werden muß.

Beide Funktionen benötigen als Parameter portid die Adresse des Ein-/Ausgabeports, also 925. Mit dem Aufruf **inportb(925)** kann das am Interface anliegende Bitmuster ausgelesen werden. Der Funktionsaufruf **outportb(925, 5)** gibt den Dezimalwert 5 aus, schaltet damit die Ausgabekanäle 0 und 2 aktiv.

Mit diesen beiden einfachen Funktionen wird die gesamte Steuerung des Interfaces abgewickelt. Allerdings muß man sich auf dieser niedrigen

Ebene selbst um die Umrechnung zwischen binären Bitmustern und dezimalen Ein-/Ausgabewerten kümmern. Um einzelne Kanäle gezielt ein- oder ausschalten zu können, lohnt sich deshalb die Definition eigener Funktionen.

Da sich die Dezimalwerte der einzelnen Kanäle aus den jeweiligen Zweierpotenzen berechnen, müssen diese ebenfalls bestimmt werden können. In Turbo-C gibt es hierfür die Bibliotheksfunktion **pow(x, y)**, welche die y. Potenz von x errechnet. Diese Funktion erwartet jedoch double-Werte für x und y und liefert ebenfalls einen double-Wert zurück. Für unsere Zwecke genügt auch eine einfache, selbst geschriebene Funktion **potenz(int b, int h)** zur Verarbeitung von Integer-Werten, deren Besonderheit in ihrer rekursiven Struktur liegt:

```
unsigned int potenz(int b, int h) /*b...Basis, h...Hochzahl*/
{
     if( h==0 ) return(1);
     else return( b * potenz(b, --h);
}
```

Für eine präzise Zeitsteuerung ist die Funktion **warte(long unsigned time)** gedacht. Sie wartet lediglich die im Parameter time angegebene Anzahl von Millisekunden, bevor sie die Programmablaufsteuerung wieder an die aufrufende Funktion zurückgibt. Um Hardwareunterschiede in der Taktgeschwindigkeit einzelner Computer auszuschließen, wird hierbei direkt der eingebaute Timer programmiert. Wer sich dafür näher interessiert, wird in der Zeitschrift c't vom April 1988 ab Seite 196 fündig. Übrigens enthält Turbo-C eine ganz ähnliche Funktion namens **delay(unsigned milliseconds)**, die hier ebenfalls Verwendung finden kann.

```
void warte(long unsigned time)
{
     int i, ticontrol = 67, ti2 = 66, pio_b = 97, pio_c;

     if( (unsigned char)peekb(0xf000,0xfffe) == 0xfc)
          pio_c = 97;        /*AT*/
     else
          pio_c = 98;        /*PC, XT*/
     outportb(ticontrol,176);
     for(i=0; i<time; i++)
     {
          outportb(ti2,169);
          outportb(ti2,4);
          outportb(pio_b,(inport(pio_b)|1));
          while((inportb(pio_c)&32) == 0);
     }
}
```

6.2.2 Ausgabefunktionen

Mit der Funktion **ein(k_nr)** soll definiert einer der Ausgabekanäle 0 bis 5 eingeschaltet werden, die Nummer des gewünschten Kanals wird dabei als Aufrufparameter übergeben. Da nur die Kanäle 0 bis 5 als Ausgabekanäle zur Verfügung stehen, sollte die Funktion zuerst die Zulässigkeit des übergebenen Parameters überprüfen und bei einer Bereichsüberschreitung als Fehlerkennzeichen den Wert 1 zurückliefern. Liegt der Parameter jedoch im zulässigen Bereich, wird der angegebene Kanal aktiviert, und zwar ohne den Zustand der anderen Ausgabekanäle zu verändern. Deshalb muß zuerst der aktuelle Dateninhalt des Ports gelesen und in diesem Bitmuster gezielt das dem gewünschten Kanal entsprechende Bit gesetzt werden. Zum Setzen eines Bits kann die logische Funktion OR verwendet werden:

Eingabe		OR
0	0	0
0	1	1
1	0	1
1	1	1

Die Wahrheitstafel zeigt, daß das Ergebnis einer OR-Verknüpfung immer dann 1 ist, wenn mindestens ein Eingangswert 1 ist. Um beispielsweise Kanal 3 einzuschalten, ohne die anderen Kanäle zu verändern, muß zuerst der aktuelle Inhalt des Interfaceports gelesen werden.

Die Funktion inportb(925) liefere den Dezimalwert 178,
der folgendem Bitmuster entspricht: 1 0 1 1 **0** 0 1 0

Dazu wird bitweise der Bitwert des Kanals 3, also $2^3 = 8$ OR 0 0 0 0 **1** 0 0 0

mit der OR-Funktion verknüpft: 1 0 1 1 **1** 0 1 0

Der über diese Verknüpfung erhaltene Datenwert wird schließlich über die Funktion **outportb()** ans Interface gesandt. Der Operator für die

bitweise OR-Verknüpfung ist in C ein senkrechter Strich "|". Diesen Strich erhalten Sie über die Tastenkombination **<Alt>124**. Dazu muß die **<Alt>**-Taste gedrückt und gedrückt gehalten werden, bis nacheinander die Zifferntasten **<1>**, **<2>** und **<4>** auf dem Numerik-Block der Tastatur gedrückt wurden. Sollen zwei Werte nicht bitweise, sondern als Gesamtwerte verknüpft werden, so schreibt man zwei senkrechte Striche "||". Diese Art der OR-Verknüpfung wird bei der Abfrage nach den zulässigen Parametergrenzen verwendet:

```
int ein(int k_nr)
{
     if(k_nr < 0 || k_nr > 5)
          return(1);
     outportb(925, inportb(925) | potenz(2, k_nr) );
     return(0);
}
```

Auf eine ähnliche Art und Weise kann die Funktion **aus(k_nr)** definiert werden, welche den über k_nr angegebenen Ausgabekanal ausschaltet, ohne die Zustände der anderen Kanäle zu verändern. Für das Ausblenden einzelner Bits wird wieder die logische Verknüpfung AND verwendet. Ihr Gebrauch soll gleich anhand eines Beispieles demonstriert werden, wobei Kanal 3 ausgeschaltet wird:

Die Funktion inportb(925) liefere den Dezimalwert 186, der folgendem Bitmuster entspricht:		1 0 1 1 **1** 0 1 0
Dazu wird bitweise der komplementäre Bitwert des Kanals 3, also NOT(2^3)=247	AND	1 1 1 1 **0** 1 1 1
mit der AND-Funktion verknüpft:		1 0 1 1 **0** 0 1 0

Zum Invertieren dient die logische Funktion NOT:

Eingabe	NOT
1	0
0	1

In C ist der Operator für eine bitweise AND-Verknüpfung das Zeichen "&", das Zeichen "~" ermöglicht ein bitweises Invertieren. Die Tilde "~" erhalten Sie über **<Alt>126**, wobei wieder die Zifferntasten auf dem Numerik-Block der Tastatur verwendet werden müssen. Die gewünschte Funktion läßt sich folgendermaßen definieren:

```
int aus(int k_nr)
{
     if(k_nr < 0 || k_nr > 5) return(1);
     outportb(925, inportb(925) & ~potenz(2, k_nr) );
     return(0);
}
```

Für manche Anwendungen ist vielleicht eine Funktion **ausgabe(int k_nr, int zustand)** sinnvoll, welche die beiden obigen Funktionen so in sich vereint, daß beim Aufruf neben der Kanalnummer noch der gewünschte Zustand als Bitwert 1 oder 0 für "ein" oder "aus" übergeben wird:

```
int ausgabe(int k_nr, int zustand)
{
     if(k_nr < 0 || k_nr > 5 || (zustand != 1 && zustand != 0) )
          return(1);
     if(zustand)
          outportb(925, inportb(925) | potenz(2, k_nr) );
     else
          outportb(925, inportb(925) & ~potenz(2, k_nr) );
     return(0);
}
```

Auf dieser niedrigen Bit-Ebene lassen sich logische Verknüpfungen sinnvoll einsetzen. Mit der Verknüpfung EXOR läßt sich schließlich noch eine Funktion realisieren, die den Zustand des angegebenen Kanals bei jedem Aufruf umschaltet. Aus der Wahrheitstafel wird ersichtlich, daß das Ergebnis von EXOR nur dann 1 ist, wenn beide Eingabewerte unterschiedlich sind:

Eingabe		EXOR
0	0	0
0	1	1
1	0	1
1	1	0

In einem Beispiel erkennen wir, wie damit ein einzelnes Bit jeweils invertiert werden kann:

```
          1 0 1 0 1 1 0 0          1 0 1 0 0 1 0 0
EXOR      0 0 0 0 1 0 0 0    EXOR  0 0 0 0 1 0 0 0
-----------------------------  -----------------------------
          1 0 1 0 0 1 0 0          1 0 1 0 1 1 0 0
```

Der Operator für die EXOR-Verknüpfung ist das Zeichen "^" und wir definieren folgende Funktion:

```
int toggle(int k_nr)
{
     if(k_nr < 0 || k_nr > 5)
          return(1);
     outportb(925, inportb(925) ^ potenz(2, k_nr) );
     return(0);
}
```

6.2.3 Eingabefunktionen

Für die Zustandsabfrage eines bestimmten Kanals muß das zugehörige Bit des Interfaceports ausgewertet werden. Das notwendige Vorgehen wurde bereits bei der BASIC-Programmierung erläutert und muß nun nur noch in eine entsprechende Funktion **eingabe(int k_nr)** übersetzt werden:

```
int eingabe(int k_nr)
{
     if(k_nr < 0 || k_nr > 7)
          return(-1);
     if(inportb(925) & potenz(2, k_nr) )
          return(1);
     else
          return(0);
}
```

Da für einen aktivierten Kanal eine 1 zurückgeliefert wird, bricht die Funktion im Fehlerfall mit dem Rückgabewert -1 ab. Bitte beachten Sie, daß als Aufrufparameter alle Werte von 0 bis 7 akzeptiert werden. Dies bedeutet, daß nicht nur die explizit als Eingabeleitungen festgelegten Kanäle 6 und 7 abgefragt werden können. Bei einem Aufruf mit einer Kanalnummer zwischen 0 und 5 wird ermittelt, ob der angegebene Ausgabekanal im Moment aktiviert ist oder nicht.

6.2.4 Funktionen zum Zählen

Die in LEGO Lines vorhandene Anweisung ZÄHLE kann in Turbo-C relativ leicht nachgebildet werden. Die hier vorgeschlagene Funktion **zaehle(int k_nr, int anzahl)** zählt die am angegebenen Eingabekanal auftretenden Aktiv-Impulse. Sobald die im zweiten Parameter angegebene Anzahl erreicht ist, wird die Steuerung an die aufrufende Funktion zurückgegeben:

```
int zaehle(int k_nr, int anzahl)
{
     int i;

     if(k_nr != 6 && k_nr != 7)
          return(1);
     for(i=0; i<anzahl; i++)
     {
          while( inport(PORT) & potenz(2, k_nr) );
          while( !(inport(PORT) & potenz(2, k_nr)) );
     }
     return(0);
}
```

Die unter LEGO Lines vorhandene Beschränkung, die von einem Eingabekanal gelieferten Impulse nicht aufsummieren und als Zahlenwert an eine Variable übergeben zu können, läßt sich in C relativ einfach beheben. Es soll deshalb eine Funktion realisiert werden, welche die nach ihrem Aufruf aufgetretenen Eingangsimpulse des angegebenen Eingabekanals aufsummiert und an ihrem Ende diese Summe der aufrufenden Funktion zurückliefert. Als Ende-Bedingung dieser Zählfunktion kommen grundsätzlich drei Möglichkeiten in Betracht:

1. Die Funktion wird durch einen beliebigen Tastendruck beendet.

2. Die Funktion wird durch Aktivieren des jeweils anderen Sensors beendet.

3. Die Funktion wird automatisch beendet, wenn seit dem letzten Aktivieren des zu zählenden Eingabekanals eine frei wählbare Zeitspanne (in Sekunden) verstrichen ist.

Deshalb wurden insgesamt drei Funktionen erstellt, die sich lediglich in ihrer Abbruchbedingung unterscheiden:

1. Funktion zaehl_t(int k_nr), Abbruch nach Tastendruck:

```
int zaehl_t(int k_nr)
{
     int zaehler=0;
     unsigned char flag=0, altflag=0;

     if(k_nr < 5 || k_nr > 7)
          return(-1);

     while(!bioskey(1))
     {
          if( inportb(925) & potenz(2,k_nr) ) flag = 1;
          if( !(inportb(925) & potenz(2,k_nr)) ) flag = 0;
          if(flag==1 & altflag==0)
               zaehler++;
          altflag = flag;
     }
     bioskey(0);
     return(zaehler);
}
```

2. Funktion zaehl_s(int k_nr), Abbruch nach Sensoraktivierung:

```
int zaehl_s(int k_nr)
{
     int zaehler=0, xk_nr;
     unsigned char flag=0, altflag=0;

     if(k_nr < 5 || k_nr > 7)
          return(-1);

     xk_nr = (k_nr == 6)? 7 : 6;

     while(!(inportb(PORT) & potenz(2, xk_nr)) )
     {
          if( inportb(925) & potenz(2,k_nr) ) flag = 1;
          if( !(inportb(925) & potenz(2,k_nr)) ) flag = 0;
          if(flag==1 & altflag==0)
               zaehler++;
          altflag = flag;
     }
     return(zaehler);
}
```

3. Funktion **zaehl_z(int k_nr, int sec)**, Abbruch nach Zeitspanne:

```
int zaehl_z(int k_nr, int sec)
{
     int zaehler=0;
     unsigned char flag=0, altflag=0;
     long zeit, w_zeit;

     if(k_nr < 5 || k_nr > 7)
     return(-1);

     w_zeit = time(&zeit) + sec;

     while(time(&zeit) < w_zeit)
     {
          if( inportb(925) & potenz(2,k_nr) ) flag = 1;
          if( !(inportb(925) & potenz(2,k_nr)) ) flag = 0;
          if(flag==1 & altflag==0)
          {
               zaehler++;
               w_zeit = time(&zeit) + sec;
          }
          altflag = flag;
     }
     return(zaehler);
}
```

6.2.5 Einbinden der Funktionen

Alle hier aufgeführten Funktionen sind als Quelltext auf der zum Buch erhältlichen Diskette für IBM PC's und Kombatible in der Datei **legofunc.h** zusammengefaßt. Um in eigenen Programmen die Funktionen:

```
int ein(int k_nr);
int aus(int k_nr);
int ausgabe(int k_nr, int zustand);
int toggle(int k_nr);
int eingabe(int k_nr);
int zaehle(int k_nr, int anzahl);
int zaehl_t(int k_nr);
int zaehl_s(int k_nr);
int zaehl_z(int k_nr, int sec);
unsigned int potenz(int b, int h);
void warte(long unsigned time);
```

aufrufen zu können, müssen Sie lediglich am Anfang Ihres C-Programms die Präprozessoranweisung **#include <legofunc.h>** aufnehmen. Die gezeigte Schreibweise setzt jedoch voraus, daß Sie zuvor die Datei legofunc.h in das INCLUDE-Verzeichnis von Turbo-C kopiert haben. Ist dies nicht der Fall, so müssen Sie den vollständigen Suchpfad angeben und in Anführungsstriche einschließen, beispielsweise:

```
#include "a:\turboc\legofunc.h"
```

Die Datei legofunc.h bindet ihrerseits die benötigten Headerdateien stdio.h, time.h, dos.h und bios.h ein. Außerdem wird die Portadresse über die Konstante PORT auf den Wert 925 festgelegt. Dies hat den Vorteil, daß eine Adreßänderung, beispielsweise beim Auftrennen des Adreßjumpers auf der Interfacekarte, problemlos möglich ist:

```
#define PORT 925    /*Bei geöffneter Adreßbrücke: 926 */
```

Alle aufgeführten Funktionen sind als Anregung für weitere Versuche gedacht und können natürlich jederzeit für eigene Anwendungen verändert und optimiert werden. Weitere Anregungen enthält das Programm **legoctrl.c**, welches eine Direktsteuerung des Interfaces über die Tastatur ermöglicht. Die einzelnen Kanäle werden auf dem Bildschirm dargestellt und die Ausgabekanäle können, wie von LEGO Lines gewohnt, direkt über die Tastatur verändert werden. Die Steuerung der Laufrichtung von an den Buchsen A, B oder C angeschlossenen Motoren kann zudem schnell über Pfeiltasten erfolgen.

Für die Eingabekanäle 6 und 7 wird nicht nur deren aktueller Zustand angezeigt: Durch einen Druck auf die entsprechende Zifferntaste wird ein Zähler gestartet, der die vom Interface kommenden Impulse am Bildschirm hochzählt. Ein Druck auf die Leertaste beendet die Zählung. Dadurch kann beispielsweise bei Modellen mit Schrittsteuerung die jeweils benötigte Schrittezahl direkt ermittelt und muß nicht mehr durch Probieren gefunden werden.

Dieses Programm enthält damit bereits einige Möglichkeiten, die der LEGO TC Controller mit seiner Direktsteuerung bietet. Dieser LEGO TC Controller mit der Schnittstelle zu Turbo Pascal wird im zweiten Teil des Buches ausführlich beschrieben werden.

6.3 Programmierung in COMAL

6.3.1 Grundlagen

Um im schulischen Bereich die mehrfach genannten Schwächen von BASIC zu überwinden, kommt hier gelegentlich die Programmiersprache COMAL zum Einsatz. Im letzten Abschnitt dieses Kapitels sollen deshalb noch einige Hinweise und Beispiele zur Interface-Programmierung unter UNICOMAL Version 2.2 gegeben werden. Etwa zehn Jahre jünger als BASIC hat COMAL eine ähnliche Syntax, verfügt aber über leistungsfähigere Kontrollstrukturen, die an PASCAL erinnern. Obwohl relativ leicht erlernbar, werden für das Verständnis der folgenden Erläuterungen grundlegende Erfahrungen in COMAL vorausgesetzt.

Dem Kenner wird bereits aufgefallen sein, daß der eigentliche Befehlsumfang von COMAL keine Anweisungen zum Datentransfer über Ein- und Ausgabeports beinhaltet. Erst über das Softwarepaket system stehen die benötigten Befehle zur Verfügung. Bevor aus COMAL auf das LEGO Interface zugegriffen werden kann, muß deshalb zuerst mit

USE system <Return>

dieses Paket aktiviert werden. Für die Ein-/Ausgabekommunikation stehen dann die Funktion

inp#(port#)

zum Einlesen und die Prozedur

out(port#, wert#)

für die Ausgabe eines Datums vom bzw. zum angegebenen Port zur Verfügung. Der Parameter port# muß in beiden Anweisungen natürlich den Wert 925 annehmen. Im Direktmodus können so beispielsweise über die Prozedur **out(925, 3)** die Ausgabekanäle 0 und 1 aktiviert werden. Die Anweisung **print inp#(925)** liefert das am Interface anliegende Bitmuster als Dezimalzahl. Der Gebrauch und die Syntax dieser Anweisungen ist damit ganz ähnlich den C-Funktionen outportb() bzw. inportb() und den BASIC-Befehlen INP und OUT. Andere Möglichkeiten sind nicht verfügbar, für etwas mehr Komfort kann man jedoch die Möglichkeiten von COMAL zur Eigendefinition von Prozeduren und Funktionen sinnvoll nutzen.

6.3.2 Prozeduren und Funktionen

Es soll ein Weg aufgezeigt werden, hilfreiche Funktionen und Prozeduren in COMAL selbst zu definieren. Da sie dieselben Aufgaben erledigen werden, wie die im vorigen Abschnitt erstellten C-Funktionen, können die bereits dort behandelten Grundlagen hier ausgespart bleiben. Außerdem liegen beiden Implementationen genau dieselben Algorithmen zugrunde, was nicht nur einen einfachen Vergleich ermöglicht, sondern auch Portierungen in weitere Sprachen erleichtert.

Da bei jedem Aufruf unserer selbstdefinierten Anweisungen überprüft werden muß, ob die als Parameter übergebene Kanalnummer einen gültigen Wert besitzt, definieren wir zunächst eine Funktion, welche diese Überprüfung erledigt:

```
0100 FUNC kanalnr_ok(untergrenze#,obergrenze#,kanalnr#)
0110   IF kanalnr#<u_grenze# OR kanalnr#>obergrenze# THEN
0120     PRINT "ungültige Kanalnummer"
0130     RETURN FALSE
0140   ELSE
0150     RETURN TRUE
0160   ENDIF
0170 ENDFUNC kanalnr_ok
```

Als Parameter werden dieser Funktion die zulässige obere und untere Grenze sowie die gewünschte Kanalnummer übergeben. Liegt diese Nummer innerhalb der angegebenen Grenzen, wird der Wahrheitswert **TRUE** zurückgegeben, sonst, nach einer entsprechenden Meldung, der Wert **FALSE**.

Da COMAL keine Potenzfunktion, wohl aber die Möglichkeit des rekursiven Funktionsaufrufes besitzt, übertragen wir die im vorigen Abschnitt vorgestellte C-Funktion potenz(). Damit sind wir in der Lage, die benötigten Wertigkeiten der einzelnen Kanäle berechnen zu lassen.

```
0190 FUNC potenz#(basis#,hochzahl#)
0200   IF hochzahl#=0 THEN
0210     RETURN 1
0220   ELSE
0230     RETURN basis#*potenz#(basis#,(hochzahl#-1))
0240   ENDIF
0250 ENDFUNC potenz#
```

Die Funktion potenz# erwartet als Parameter die Basis und den Exponenten als ganze Zahl und liefert das Funktionsergebnis ebenfalls als ganze Zahl zurück.

Das Aktivieren eines gewünschten Kanals geschieht über die Prozedur:

```
0270 PROC ein(kanalnr#)
0280   IF kanalnr_ok(0,5,kanalnr#) THEN
0290     out(925,inp#(925) BITOR potenz#(2,kanalnr#))
0300   ENDIF
0310 ENDPROC ein
```

Zunächst wird die als Prozedurparameter eingegebene Kanalnummer von der Funktion **kanalnr_ok** auf zulässige Werte untersucht. Diese Überprüfung findet am Anfang jeder Prozedur oder Funktion statt und wird bei den folgenden Beispielen deshalb nicht mehr explizit erwähnt.

Liegt der Wert in den zulässigen Grenzen, so wird das momentan am Interface anliegende Bitmuster über inp#(925) eingelesen und mit dem über die Potenzfunktion errechneten Dezimalwert der Kanalnummer bitweise "verODERt". Die Ausgabe dieses Ergebnisses aktiviert den gewünschten Kanal, ohne andere Kanäle zu beeinflussen.

Ausgeschaltet wird ein Kanal mit der Prozedur:

```
0330 PROC aus(kanalnr#)
0340   IF kanalnr_ok(0,5,kanalnr#) THEN
0350     out(925,inp#(925) BITAND (255-potenz#(2,kanalnr#)))
0360   ENDIF
0370 ENDPROC aus
```

Da in COMAL eine bitweise Negation fehlt, muß das Invertieren hier über eine Subtraktion der Kanalwertigkeit von der Zahl 255 (entspricht dem Bitmuster 1111 1111) erfolgen. Diese Differenz wird mit dem aktuell am Interface anliegenden Bitmuster über die logische Funktion UND verknüpft. Die Ausgabe dieses Ergebnisses schaltet dann den gewünschten Kanal aus.

```
0390 PROC ausgabe(kanalnr#,zustand#)
0400   *IF kanalnr_ok(0,5,kanalnr#) THEN
0410     IF zustand#=1 THEN
0420       out(925,inp#(925) BITOR potenz#(2,kanalnr#))
0430     ELIF zustand#=0 THEN
0440       out(925,inp#(925) BITAND(255-potenz#(2,kanalnr#)))
0450     ELSE
0460       PRINT "ungültiger Kanalzustand"
0470     ENDIF
0480   ENDIF
0490 ENDPROC ausgabe
```

Die Prozedur **ausgabe** vereint die Prozeduren **ein** und **aus** derart in sich, daß beim Aufruf neben der Kanalnummer als zweiter Parameter dessen gewünschter Zustand mit angegeben werden kann. Dabei kann der Parameter zustand# die Werte 1 und 0 zum Ein- und Ausschalten annehmen.

Umgeschaltet wird ein Kanal mit der folgenden Prozedur:

```
0510 PROC toggle(kanalnr#)
0520   IF kanalnr_ok(0,5,kanalnr#) THEN
0530     out(925,inp#(925) BITXOR potenz#(2,kanalnr#))
0540   ENDIF
0550 ENDPROC toggle
```

Für das Umschalten ist die logische Verknüpfung XOR verantwortlich. Ihre Wertetafel finden Sie ebenfalls im vorhergehenden Abschnitt.

Zum Einlesen eines Wertes muß eine Funktion definiert werden, da Prozeduren keine Rückgabewerte überliefern können:

```
0570 FUNC eingabe#(kanalnr#)
0580   IF kanalnr_ok(0,7,kanalnr#) THEN
0590     IF (inp#(925) BITAND potenz#(2,kanalnr#)) THEN
0600       RETURN 1
0610     ELSE
0620       RETURN 0
0630     ENDIF
0640   ENDIF
0650 ENDFUNC eingabe#
```

Wenn der angegebene Kanal gerade aktiv ist, wird der Wert 1 zurückgegeben, sonst der Wert 0.

Die LEGO Lines Strukturanweisung ZÄHLE wird in der folgenden Prozedur nachgebildet:

```
0670 PROC zähle(kanalnr#,anzahl#)
0680   IF kanalnr_ok(6,7,kanalnr#) THEN
0690     FOR zähler#:=1 TO anzahl# DO
0700       WHILE (inp#(925) BITAND potenz#(2,kanalnr#)) DO NULL
0710       WHILE NOT (inp#(925) BITAND potenz#(2,kanalnr#)) DO NULL
0720     ENDFOR zähler#
0730   ENDIF
0740 ENDPROC zähle
```

Die im zweiten Parameter angegebene Anzahl von Sensoraktivierungen wird als Endwert einer FOR-Schleife verwendet.

Zum Zählen der Sensoraktivierungen werden wieder drei ähnliche Funktionen vorgestellt, die sich in ihrer Abbruchbedingung unterscheiden:

1. Abbruch nach Tastendruck:

Diese Funktion liefert die nach ihrem Aufruf bis zum Betätigen einer beliebigen **Taste der Computertastatur** erfolgten Aktivierungen des angegebenen Eingabekanals als Funktionsergebnis zurück:

```
0760 FUNC zähle_t#(kanalnr#)
0770   IF kanalnr_ok(6,7,kanalnr#) THEN
0780     zähler#:=0; flag#:=0; altflag#:=0
0790     WHILE KEY$="" DO
0800       IF (inp#(925) BITAND potenz#(2,kanalnr#)) THEN flag#:=1
0810       IF NOT (inp#(925) BITAND potenz#(2,kanalnr#)) THEN flag#:=0
0820       IF flag#=1 AND altflag#=0 THEN zähler#:+1
0830       altflag#:=flag#
0840     ENDWHILE
0850     RETURN zähler#
0860   ELSE
0870     RETURN 0
0880   ENDIF
0890 ENDFUNC zähle_t#
```

2. Abbruch nach Sensorbetätigung:

Diese Funktion liefert die nach ihrem Aufruf bis zum Betätigen des jeweils anderen **Sensors** erfolgten Aktivierungen des angegebenen Eingabekanals als Funktionsergebnis zurück:

```
0910 FUNC zähle_s#(kanalnr#)
0920   IF kanalnr_ok(6,7,kanalnr#) THEN
0930     IF kanalnr#=6 THEN
0940       xkanalnr#:=7
0950     ELSE
0960       xkanalnr#:=6
0970     ENDIF
0980     zähler#:=0; flag#:=0; altflag#:=0
0990     WHILE NOT (inp#(925) BITAND potenz#(2,xkanalnr#)) DO
1000       IF (inp#(925) BITAND potenz#(2,kanalnr#)) THEN flag#:=1
1010       IF NOT (inp#(925) BITAND potenz#(2,kanalnr#)) THEN flag#:=0
1020       IF flag#=1 AND altflag#=0 THEN zähler#:+1
1030       altflag#:=flag#
1040     ENDWHILE
1050     RETURN zähler#
1060   ELSE
1070     RETURN 0
1080   ENDIF
1090 ENDFUNC zähle_s#
```

3. Abbruch nach Zeitspanne:

Diese Funktion wird beendet, wenn seit dem letzten Aktivieren des angegebenen Eingabekanals die als zweiter Parameter angegebene **Zeitspanne** (in Sekunden) verstrichen ist. Zurückgeliefert wird die Anzahl der bis zu diesem Zeitpunkt erfolgten Aktivierungen:

```
1110 FUNC zähle_z#(kanalnr#,zeit#)
1120   IF kanalnr_ok(6,7,kanalnr#) THEN
1130     zähler#:=0; flag#:=0; altflag#:=0
1140     TIMER 0
1150     WHILE TIMER<zeit# DO
1160       IF (inp#(925) BITAND potenz#(2,kanalnr#)) THEN flag#:=1
1170       IF NOT (inp#(925) BITAND potenz#(2,kanalnr#)) THEN flag#:=0
1180       IF flag#=1 AND altflag#=0 THEN
1190         zähler#:+1
1200         TIMER 0
1210       ENDIF
1220       altflag#:=flag#
1230     ENDWHILE
1240     RETURN zähler#
1250   ELSE
1260     RETURN 0
1270   ENDIF
1280 ENDFUNC zähle_z#
```

Natürlich brauchen Sie diese Funktionen nicht mühsam abzutippen, auf der Begleitdiskette im MS-DOS-Format sind alle Funktionen und Prozeduren im Programmfile **legofunc.cml** zusammengefaßt. Nach dem Laden und Starten dieser Datei haben Sie auch im Direktmodus von COMAL Zugriff auf alle aufgeführten Anweisungen.

6.3.3 Das Modul LEGOCML.PKG

Über das Einbinden von Softwaremodulen steht in COMAL eine weitere komfortable Möglichkeit des Zugriffs auf benutzerdefinierte Funktionen zur Verfügung. Auf der Begleitdiskette im MS-DOS-Format befindet sich deshalb auch ein File **legocml.pkg**, mit welchem die folgenden Prozeduren und Funktionen eingebunden werden können:

PROC ein (kanalnr#)
PROC aus (kanalnr#)
PROC ausgabe (kanalnr#, zustand#)
PROC toggle (kanalnr#)

PROC zaehle (kanalnr#, anzahl#)

FUNC eingabe# (kanalnr#)
FUNC zaehle_s# (kanalnr#)
FUNC zaehle_t# (kanalnr#)
FUNC zaehle_z# (kanalnr#, zeit#)

Im Gebrauch ergeben sich keine Unterschiede zu den zuvor erstellten Prozeduren und Funktionen, lediglich die Funktion **zaehle_z** besitzt eine höhere Auflösung des Zeitintervalles und erwartet die Angabe der Zeitspanne in Zehntelsekunden.

Der Vorteil der Prozedureinbindung über Softwaremodule besteht in deren völliger Transparenz. Es braucht kein COMAL-Programmcode in den Speicher geladen zu werden, da direkt ausführbarer Code mit dem COMAL-System verbunden wird. Die enthaltenen Anweisungen erweitern damit direkt den Sprachumfang von COMAL.

Wenn Sie das Modul **legocml.pkg** von der Begleitdiskette in Ihr COMAL-Verzeichnis kopiert haben, müssen Sie nur noch mit der Anweisung

LINK "legocml <Return>

das Softwarepaket an COMAL anbinden und mit

USE lego <Return>

aktivieren. Ab sofort stehen Ihnen dann die oben aufgeführten Prozeduren und Funktionen in COMAL zur Verfügung.

Die zusätzlichen Softwaremodule für COMAL müssen in Maschinensprache erstellt werden. Assemblerfreaks finden deshalb auf der Diskette auch den Quellcode **legocml.asm**, der einerseits das Einbinden von eigenen Modulen in COMAL und andererseits die Interface-Programmierung auf unterster Maschinenebene zeigt. Hier im Buch wollen wir jedoch aus diesen Tiefen rasch wieder aufsteigen und im folgenden zweiten Teil eine komfortablere Benutzeroberfläche, den LEGO TC Controller, vorstellen.

7 Der LEGO TC Controller

Ohne spezielle Anwendersoftware erfordern Steuerungs- und Regelungsaufgaben eine intime Kenntnis der Hardware. Welches sind die Adressen und Bitpositionen der einzelnen Leitungen, und welche Eigenschaften hat das Interface? Diese Daten sind von Computertyp zu Computertyp verschieden. Hinzu kommt das notwendige Wissen, wie man in der verwendeten Programmiersprache auf die Ein- und Ausgabeleitungen Zugriff nehmen kann. Im vorangehenden Kapitel wurden hierzu Beispiele gebracht. Dabei sind auch eventuell Unterschiede in den verschiedenen Dialekten einer Programmiersprache zu beachten.

Unter diesen Gegebenheiten stellte die Benutzeroberfläche des Programms LEGO Lines einen enormen Fortschritt dar, was dann ja auch mit der Verleihung des Schul-Software-Preises des Jahres 1987 Anerkennung gefunden hat. Leider ist LEGO Lines aber nur auf zwei Computertypen implementiert: den Homecomputern Commodore 64 und 128 und den IBM-PC's und den dazu kompatiblen MS-DOS Computern. Die Benutzeroberfläche ist bei beiden Typen nahezu gleich. Die bestehenden gewaltigen Hardwareunterschiede brauchen den Anwender überhaupt nicht zu interessieren.

LEGO Lines hat als reine Schulsoftware jedoch auch recht enge Grenzen. Einmal werden diese gebildet durch die wenigen Strukturanweisungen, mit denen Prozeßabläufe programmiert werden können, und dann ist es das Fehlen von Variablenspeichern. Verzweigte Programmabläufe sind nur unmittelbar bei einer Eingabeportabfrage möglich. Weiter kann auch die Geschwindigkeit der Abarbeitung der Steuerungs- und Regelungsprogramme in LEGO Lines ein Handicap sein, welches durch den Ablaufmodus beim IBM-PC und Kompatiblen zwar abgemildert ist.

Der LEGO TC Controller stellt hier zwischen der direkten Programmierung und der reinen Anwenderlösung einen gelungenen Mittelweg dar. Der TC Controller bindet mittels der leistungsfähigen und weit verbreiteten Programmiersprache Pascal (Turbo Pascal und Quick Pascal) die LEGO Hardware ein in die allgemeine Lösung von Steuerungs- und Regelungsproblemen. Mit dem TC Controller sind dem Anwender Prozeduren und Funktionen verfügbar, die die Möglichkeiten der Hardware voll ausnützen, dem Anwender aber keine speziellen Hardwarekenntnisse abfordern.

7.1 Das Anfertigen einer Arbeitsdiskette

Vor dem ersten Start des LEGO TC Controllers fertigen wir uns zweckmäßigerweise eine Arbeitsdiskette an. Mit dieser Diskette im Laufwerk A: werden wir dann jedesmal den Computer einschalten. Dies stellt sicher, daß das LEGO TC Controller-Programm nicht mit anderen bereits im Arbeitsspeicher des Computers befindlichen Programmen in Konflikt gerät.

Wir beschreiben dazu kurz die mit der DOS-Version 3.3 notwendigen Arbeiten. Gegenüber früheren DOS-Versionen gilt es hier etwas andere Gegebenheiten zu beachten. Wir nehmen an, daß unser Computer zwei Diskettenlaufwerke besitzt. Unsere noch leere Arbeitsdiskette befinde sich im Laufwerk A: und in B: die Diskette mit den DOS-Betriebssystemdateien.

Wir gehen weiter von einer 5,25" Diskette mit einer Kapazität von 360 KByte aus, die auch die zum Starten - dem sog. 'Booten' - notwendigen DOS-Dateien erhalten soll. Eine solche Diskette wird beim Formatieren mit der FORMAT-Kommandodatei mit dem Parameter /s erzeugt:

```
A>b:format a: /s
```

Befindet sich dann auf dieser Arbeitsdiskette nach dem Formatieren die Kommandodatei **COMMAND.COM**, dürfen wir sicher sein, daß auch die notwendigen beiden anderen, jedoch nicht sichtbaren DOS-Dateien vorhanden sind.

Damit nun beim Starten mit dieser Diskette unser Computer auch auf seinen Standort in Deutschland eingestellt werden kann, müssen wir noch die DOS-Dateien **COUNTRY.SYS**, **KEYBOARD.SYS** und **KEYB.COM** auf die Arbeitsdiskette kopieren. Diese Dateien werden vom Laufwerk B: von der Diskette mit den Betriebssystemdateien mit

```
copy b:country.sys a:     und
copy b:key*.* a:
```

auf die Arbeitsdiskette im Laufwerk A: kopiert.

Besitzt Ihr Computer eine Festplatte C:, können diese DOS-Dateien auch von dort auf die Arbeitsdiskette kopiert werden. Es muß dann statt b: nur der vollständige Pfad mit der Festplattenbezeichnung, z. B. c:\dos\key*.*, eingegeben werden.

Das Einstellen auf Deutschland und eine deutsche Tastatur erfolgt beim Start mit den beiden Dateien **CONFIG.SYS** und **AUTOEXEC.BAT**, die wir aber erst erzeugen und auf die Arbeitsdiskette schreiben müssen.

Wir legen die Datei CONFIG.SYS an, indem wir über die Tastatur folgende (Befehls-)Zeilen eingeben:

```
copy con a:config.sys
country = 049
^Z
```

Nach der Eingabe von ^Z, das durch Drücken der Funktionstaste **<F6>** oder durch die Tastenkombination **<Ctrl>-<Z>** auf dem Bildschirm erscheint, wird die zweite Zeile mit der Telefonvorwahlnummer von Deutschland in einer Datei CONFIG.SYS auf die Diskette geschrieben: Unser Computer weiß damit, daß er uns das Datum und die Uhrzeit in der in Deutschland üblichen Form mitteilen bzw. abfragen soll.

Trotz dieser Anpassung an nationale Gegebenheiten gibt es jedoch bei Computern und nicht nur dort immer noch keine volle Kompatibilität. Wenn sie beispielsweise auf Ihrer Tastatur die Taste **<Ctrl>** vergeblich suchen, sollten Sie es mit der Taste **<Strg>** probieren. Sie hat bei gleicher Funktion nur eine (eingedeutschte) andere Bezeichnung. Ähnlich ist es bei verschiedenen anderen Tasten. Im Anhang haben wir Ihnen korrespondierende unterschiedliche Tastenbezeichnungen gegenübergestellt.

Ähnlich wie CONFIG.SYS schreiben wir die AUTOEXEC.BAT Datei:

```
copy con a:autoexec.bat
keyb gr
date
time
ver
control
^Z
```

Beachten Sie in der zweiten Zeile das Leerzeichen vor der Parameterangabe **gr**. Dieser Befehl bewirkt, daß über die vorhin kopierten KEY*.*-Dateien bei einem Tastendruck auch diejenigen Zeichen, die auf den Tasten unserer deutschen DIN-Tastatur stehen, auf dem Bildschirm sichtbar werden. Ohne diesen Befehl würde uns unsere Tastatur sonst beispielsweise ein z für ein y vormachen.

Die beiden nächsten Befehlszeilen ermöglichen uns beim Booten des Computers das Aktualisieren des Datums und der Systemzeit. Mit der vorletzten Zeile **control** haben wir unserem nächsten Tun beim Anfertigen der Arbeitsdiskette in gewisser Weise vorgegriffen. Dieser Befehl erwartet nämlich das Vorhandensein der ausführbaren Datei CONTROL.EXE auf der Start- und Arbeitsdiskette.

Bis jetzt hat sich aber unsere Vorbereitung einer Startdiskette nur auf Dateien beschränkt, die vom Betriebssystem her für ein vernünftiges Arbeiten mit dem Computer notwendig sind. Nun brauchen wir noch das Programm für den Betrieb des LEGO TC Controllers.

Dazu kopieren wir mit dem Befehl

```
copy b:control.exe a:
```

die Datei CONTROL.EXE von der LEGO TC Controller Master Disk auf unsere Arbeitsdiskette, deren Inhalt wir uns anschließend mit dir a: anzeigen lassen (Abb. 7.1):

```
A>dir a:

Diskette/Platte, Laufwerk A:, hat den
Namen LEGOCONTROL
Verzeichnis von A:\

COMMAND  COM    25979  18.03.87  12.00
COUNTRY  SYS    11285  18.03.87  12.00
KEYBOARD SYS    19766  18.03.87  12.00
KEYB     COM     9168  18.03.87  12.00
CONFIG   SYS       15  01.03.90  10.00
AUTOEXEC BAT       37  01.03.90  10.00
CONTROL  EXE    36512   7.12.89  21.25
         7 Datei(en)    201728 Byte frei
```

Abb. 7.1 Directory der Arbeitsdiskette

Die Bildschirmausgabe zeigt, daß von den 362496 Byte der formatierten Diskette noch 201728 Byte frei sind. Dies erscheint viel. Es ist jedoch zu wenig, um später bei unserer fortgeschritteneren Arbeit mit dem LEGO TC Controller auch noch die Turbo Pascal Dateien für das Programmieren in der integrierten Entwicklungsumgebung aufzunehmen. Wir werden dann dafür einen Ausweg finden müssen.

7.2 Die Direktsteuerung mit dem LEGO TC Controller

Nach all den Vorbereitungsarbeiten ist es an der Zeit, nun endlich mit dem LEGO TC Controller wirklich Bekanntschaft zu machen. Wir belassen dazu die Arbeitsdiskette im Laufwerk A: und veranlassen mit der Dreifachtastenkombination **<Ctrl>-<Alt>-<Del>** einen Warmstart des Computers: Er wird neu gebootet.

Nach dem eventuell notwendigen Aktualisieren des Datums und der Uhrzeit wird das Programm CONTROL.EXE eingelesen und sogleich ausgeführt. Der Bildschirm hat das Aussehen von Abb. 7.2.

```
A>time
Systemzeit: 21.30.00,00
Neue Zeit (hh.mm.ss) eingeben:

A>ver

IBM Personal Computer DOS-Version  3.30

A>control
```

```
LEGO TC Controller V 2.00
LEGO TC Direktsteuerung V 2.00
(C) LEGO Group 1988,1990
```

```
Die LEGO TC Direktsteuerung V 2.00 ist im Speicher installiert!
Aufruf mit <Alt> + <i>
```

```
Der LEGO TC Controller V 2.00 ist im Speicher installiert!
```

Abb. 7.2 Bildschirm nach dem Laden des LEGO TC Controllers

Machen wir uns an das Ausforschen und das Erproben dieses Programms. Aus der Bildschirmmeldung (Abb. 7.2) dürfen wir entnehmen, daß das Programm aus praktisch zwei Programmteilen besteht:

- der LEGO TC Direktsteuerung und
- dem LEGO TC Controller.

Der LEGO TC Controller ist das eigentliche Programm, welches im Hintergrund ständig und unabhängig vom Benutzer das LEGO Interface überwacht. Die Direktsteuerung stellt eine besondere Schnittstelle zwischen uns und diesem Steuerungsprogramm dar.

Wir probieren die Tastenkombination **<Alt>-<i>** aus, welche die Direktsteuerung auflegt. Der Begriff Tastenkombination gibt immer das gleichzeitige Drücken der betreffenden Tasten an. Mit **<Alt>-<i>** erscheint auf der rechten Bildschirmhälfte der nachfolgend abgebildete Direktsteuerungsrahmen und in der untersten Bildschirmzeile, der Referenzzeile, die Auflistung von Tastenfunktionen (Abb. 7.3).

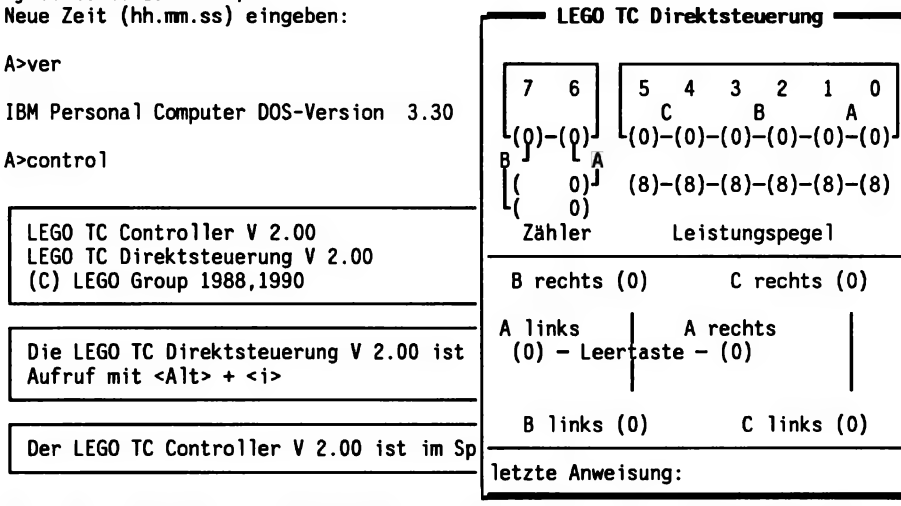


Abb. 7.3 Bildschirm mit dem Direktsteuerungsfenster

Probieren wir die erste Tastenfunktion gleich aus und drücken die **<Esc>**-Taste. Das eingeblendete Direktsteuerungsfenster verschwindet und der alte, vorher überdeckte Bildschirminhalt ist wieder voll dargestellt.

Versuchten wir nun ein erneutes Laden des CONTROL-Progamms, bekommen wir bei der Eingabe von

control

die Meldung:

Bereits installiert!

Dies zeigt an, daß sich das Programm nach dem ersten Laden dauerhaft, d. h. resident, im Arbeitsspeicher des Computers eingenistet hat. Mit **<Alt>-<i>** läßt es sich dadurch jederzeit wieder aktivieren.

Entsprechend dem Handbuch kann die Direktsteuerung aktiviert werden

- auf Betriebssystemebene, wie gerade geschehen,
- aus einem Programmeditor, z. B. in der integrierten Entwicklungsumgebung von Turbo Pascal, und auch
- aus einem laufenden Programm heraus.

Die Tastenkombination **<Alt>-<i>** ist damit ein sog. 'Hot Key'. Stellt ein Programmpaket wie z. B. Framework den 'Hot Key' ausnahmsweise "kalt", ist er nach dem Ausstieg aus dem Programmpaket wieder "heiß".

Hätten wir von der Möglichkeit des Anschlusses und der Steuerung eines zweiten Interfaces Gebrauch gemacht, könnten wir nach dem Laden des Programms CONTROL2.EXE das entsprechende Direktsteuerungsfenster dazu mit **<Alt>-<j>** auf den Bildschirm holen.

7.2.1 Direktsteuerung der Ausgänge

Beim Betrachten des Direktsteuerungsfensters (Abb. 7.3) fallen zumindest bei den obersten Zeilen Ähnlichkeiten mit dem LEGO Interface (vgl. Abb. 3.1) und dem Bildschirm des Softwarepakets LEGO Lines (vgl. Abb. 4.1) auf: In der Kopfzeile sind die 8 Kanalleitungen zum LEGO Interface dargestellt. Wir erkennen auch die Einteilung dieses Ports in 6 Ausgabe- und 2 Eingabekanäle. Bei den Ausgabekanälen bringen wir außerdem die Buchstaben A, B und C richtig mit den Anschlußmöglichkeiten auf dem LEGO Interface für die Elektromotoren aus den LEGO Technic Control Sets in Verbindung. Auch ist nicht schwer zu erraten, daß die eingeklammerten Nullen den jeweiligen Signalzustand auf den einzelnen Leitungen wiedergeben.

Lesen wir dann noch in der Erläuterungszeile für die Tastenfunktionen

0 .. 5 Ausgänge,

wissen wir Bescheid. Mit einem Drücken einer der Zifferntasten schalten wir den Signalzustand des jeweiligen Kanals um. Aus einer 0 wird eine 1 und umgekehrt. Beim Interface sehen wir den Signalzustand zusätzlich am Leuchten oder Nichtleuchten der zugeordneten roten Kontrollampen.

So einleuchtend diese Zuordnung von Zifferntasten und Portkanälen ist, so unangenehm ist die jeweils umgekehrte Reihenfolge der Kanalbezeichnungen auf dem Bildschirm, bzw. dem Interface und der Tastatur, wo die Zifferntasten von links nach rechts laufen, während die Portleitungen von rechts nach links bezeichnet sind. Einzig die Null ist seitenrichtig eingeordnet; dafür aber ist sie bezüglich der anderen Kanäle isoliert.

Gerade für eine Direktsteuerung von Prozessen wäre eine auch folgerichtige gegenseitige Zuordnung hilfreich, wenn für eine fehlbedienungsfreie Steuerung nicht sogar notwendig. Kann man sich beim Interface für den größeren Teil der Kanäle noch behelfen, indem man das Interface einfach um 180° dreht, ist man beim Bildschirm hilflos; denn den Bildschirm noch auf dem Kopf stehend zu betrachten, geht leider nicht.

Die Probleme dieser bereits bei LEGO Lines verkehrten Zuordnung werden schon einsichtig beim Versuch, z. B. das Ampelmodell direkt zu steuern. Schalten Sie dazu einfach die drei Lampen GRÜN-GELB-ROT des Ampelmodells an die Ausgabekanäle 5-4-3. Die zu schaltende Phasenfolge ist: GRÜN, GELB, ROT, ROT-GELB (vgl. Projekt 4.3). Wir schaffen das richtige Schalten von nicht zu langen Zeitintervallen fast nur, wenn wir nicht auf das Interface und den Bildschirm blicken, sondern uns fest auf das Ampelmodell konzentrieren oder noch besser auf das um 180° gedrehte Programmraster von Seite 36 (Abb. 4.8) fixiert sind und dieses dann von unten nach oben abarbeiten.

Man hätte schon in den Anfängen der Konzeption von LEGO Lines und dem TC Controller davon ablassen müssen, die Bezeichnung der Portkanäle von ihren jeweiligen Bitpositionen im Bündel der Signalleitungen des Portbytes, dem jeweiligen Stellenwert im Binärmuster, abzuleiten. Dies wäre auch konsequent gewesen, da LEGO Lines das POKEn und PEEKen von Zahlen und die Kenntnis von speziellen Hardwarevoraussetzungen ja sonst außen vor ließ.

7.2.2 Eingabekanäle auf dem LEGO Interface

Wenden wir uns nach dieser kritischen Bemerkung wieder der weiteren Ausleuchtung und Erforschung des Bildschirmfensters und der Direktsteuerung zu. Die beiden Anschlüsse 6 und 7 sind Eingabekanäle. Wir können an sie entweder einen Tastensensor- oder einen Optosensor-Baustein anschließen. Der momentan eingestellte Signalzustand der einzelnen Eingabeleitung läßt sich dann auf dem Bildschirm ablesen. Ist die zugehörige Anzeigelampe auf dem Interface dunkel, sehen wir auf der Bildschirmanzeige eine 0, eine 1 dagegen bei grün leuchtender Kontrollampe.

Nun sind die Bildschirmanalogien von LEGO Lines und der TC Direktsteuerung schon weitgehend erschöpft. Die nachfolgenden Möglichkeiten von direkten Eingriffen in Steuerungsprozesse sind Neuerungen, die uns nur die TC Direktsteuerung bietet.

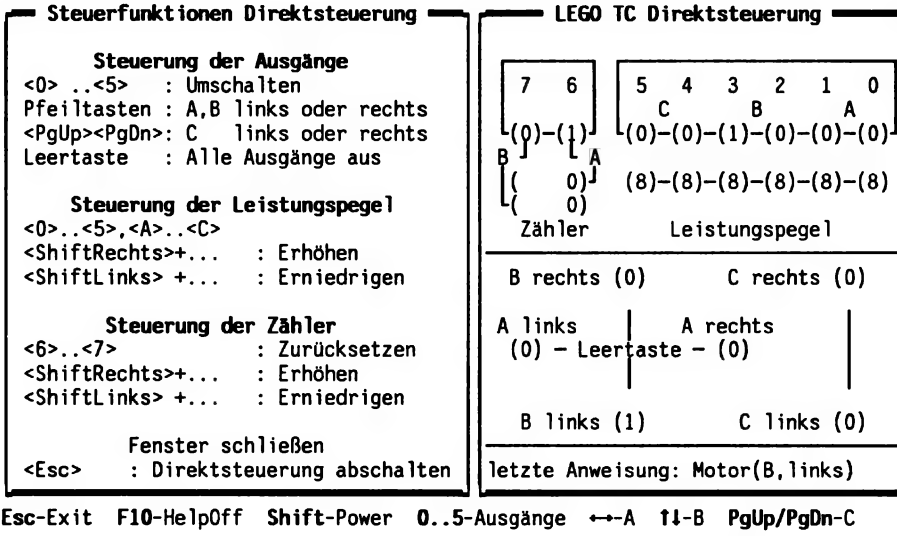


Abb. 7.4 Hilfefenster bei der Direktsteuerung

Um uns im Erforschen der Direktsteuerung auf etwas sichererem Boden zu bewegen, nehmen wir die in der untersten Zeile angebotene Hilfefunk-

tion an und bringen mit **<F10>** zusätzlich das Hilfefenster auf die linke Bildschirmhälfte (Abb. 7.4): Es enthält weitgehend die gleichen Informationen wie die unterste Bildschirmzeile mit den Tastenbefehlen; die verfügbaren Steuerungsfunktionen sind jedoch etwas ausführlicher formuliert und damit leichter verständlich.

Ein erneutes **<F10>** schaltet das Hilfefenster wieder weg. Mit **<Esc>** schaltet man beide Fenster, sowohl das Hilfefenster wie das der Direktsteuerung weg. Mit **<Alt>-<i>** kann jedoch dann der vorangehende Fensterzustand der Direktsteuerung wieder zurückgeholt werden.

7.2.3 Motorsteuerung

Als nächstes betrachten wir den mittleren Teil im Fenster der LEGO TC Direktsteuerung. Dieser Teil dient ebenso dem Umschalten von Signalzuständen, wie wir aus dem Hilfefenster für die Direktsteuerung entnehmen können: Mit den Pfeiltasten lassen sich die Motorausgänge am Interface schalten, die dort jeweils zwei Ausgangskanäle überbrücken.

Der Anfangszustand aller Ausgänge sei 0. Sie erreichen dies am schnellsten, indem Sie einfach die **<Leertaste>** kurz antippen. Damit kann man gemäß den Informationen im Hilfefenster und im Steuerungsfenster alle Ausgänge gleichzeitig ausschalten.

Drücken wir nun beispielsweise die **<→>**-Taste, so schaltet der Signalzustand des rechten Kanals des Motoranschlusses A um. Dies ist gleichbedeutend mit dem Setzen des Bits von Kanal 0. Der Schaltzustand wird gleichzeitig im oberen Fensterteil und im mittleren Teil des Steuerfensters bei den Motorsteuerfunktionen angezeigt. Ein nochmaliges Drücken der gleichen Pfeiltaste schaltet dieses Bit wieder weg. Was soll nun aber diese Motordirektschaltung, da sie bis jetzt ja noch keinen Unterschied gegenüber dem direkten Schalten mit der Taste **<0>** zeigt?.

Um den Sinn der Motordirektschaltung zu erfahren, drücken wir abwechselnd die **<→>**- und dann die **<←>**-Taste. Was fällt auf? Das Drücken einer Pfeiltaste bewirkt zweierlei: einmal schaltet sie den ihr zugeordneten Kanal um, von 0 auf 1 oder von 1 auf 0, und dann schaltet sie aber gleichzeitig den anderen Kanal auf 0 zurück, wenn dieser auf 1 geschaltet war. Mit der Motordirektsteuerung können wir so einen Motor mit nur

einem Tastendruck, jeweils abwechselnd auf die einander paarweise zugeordneten Pfeiltasten in seiner Drehrichtung umschalten. Es sind nicht wie bei der Einzelportschaltung über die Zifferntasten für ein Umschalten zwei Tastendrucke nötig. Auch ein fast gleichzeitiges Umschalten von mehreren Motoren kann dadurch sicherer und schneller erledigt werden.

Die Tasten <→> und <←> sind dem Motoranschluß A, <↑> und <↓> dem Motor B und schließlich die <**Bildauf**>- und <**Bildab**>-Taste, auch als <**PgUp**> und <**PgDn**> bezeichnet, dem Motor C zugeordnet. Die Anordnung der Tastenfunktionen im Bildschirmfenster der Direktsteuerung entspricht dabei zudem der Anordnung der jeweiligen Tasten auf der Tastatur. Unsere vorhin geäußerte Kritik bezüglich der Zifferntasten ist hier also nicht berechtigt.

Die doppelte Schaltfunktion, Umschalten eines Kanals und Zurückschalten des anderen, gilt auch, wenn beide für den Motor zuständigen Bitkanäle über die Zifferntasten auf 1 geschaltet sind; die beiden Kontrollampen sind dabei hell. Im Fensterteil der Motorsteuerung wird in diesem Fall aber trotzdem als Zustand der beiden Motorkanäle 0 angegeben. Dies bedeutet dann, daß beide Motoranschlüsse das gleiche Spannungspotential haben und der Motor darum nicht läuft.

Wird bei diesem Zustand des an beiden Motoranschlüssen anliegenden Spannungspotentials eine der beiden zuständigen Pfeiltasten gedrückt, dann werden auch die Signalzustände in den Bitkanälen auf 0 zurückgesetzt; beide Kontrollampen werden ausgeschaltet. Für den Motor selbst hat dies jedoch keine Funktion, da an ihm sowohl im einen wie im anderen Fall keine Spannung zwischen seinen beiden Anschlüssen anliegt. Methodisch ist es jedoch im Abbild zur technischen Wirklichkeit sinnvoll, wenn auch ein ungefährliches elektrisches Gerät wie unser Elektromotor im Ruhezustand keine Spannung gegen das Nullpotential besitzt.

Ohne weitere Mühe testen wir diese Schaltmöglichkeiten der Motordirektsteuerung. Wir verbinden dazu einfach einen Motor mit dem Interface. Dabei werden wir schnell die Beobachtung machen, daß von einer Bewegung bei einem Modell mehr Motivation ausgeht als von dem bloßen Aufleuchten irgendwelcher Lämpchen. Methodisch sollte man diese Eigenerfahrung dann auch im Unterricht mit Schülern umzusetzen versuchen, indem man bewegten Modellen gegenüber statischen den Vorzug gibt.

7.2.4 Leistungspegel

Für die Bewegung der Motoren wie für das Leuchten der Lampen hat die Direktsteuerung noch eine weitere Neuerung. Es läßt sich nämlich die Intensität der Bewegung bzw. des Leuchtens über den Leistungspegel einstellen. Die Anzeige für diesen sehen wir im oberen Teil des Steuerungsfensters.

Die sechs eingeklammerten Ziffern stellen mit dem Wert 8 die maximal verfügbare Leistung am jeweiligen Ausgang dar. Der Pegelwert läßt sich über Tastenkombination, d. h. dem gleichzeitigen Drücken von mehreren Tasten verändern. Dazu ist immer eine der beiden **<Shift>**-Tasten auf der Tastatur nötig. Halten wir die linke Shift-Taste, im Hilfefenster **<ShiftLinks>** bezeichnet, niedergedrückt und tippen beispielsweise kurz auch auf die Zifferntaste **<3>**, dann erniedrigt sich der auf dem Bildschirm sichtbare Wert bei jedem Drücken um 1 solange, bis in der Klammer der Wert 0 angezeigt wird. Wird statt der **<ShiftLinks>**-Taste die **<ShiftRechts>**-Taste gedrückt gehalten, erhöht sich der Pegelwert des Kanals bei jedem Druck der zugehörigen Zifferntaste, sofern der Wert nicht schon den Maximalwert 8 hat.

Ist am Interface ein Motor oder eine Lampe angeschlossen und der jeweilige Bitwert des Kanals 1, so werden wir die Bedeutung des Begriffs Leistungspegel sofort direkt erfahren. Beim Herabschalten des Leistungspegels dreht der am Kanal angeschlossene Motor immer langsamer oder die Lampe leuchtet immer weniger. Beim Leistungspegel 0 hört das Drehen des Motors und das Leuchten der Lampe dann ganz auf. Leistungspegel 0 ist also in der Wirkung gleichbedeutend mit dem Bitwert 0 am betreffenden Kanal.

Ein eingestellter Pegelwert zeigt immer nur Wirkung, wenn auch das betreffende Kanalbit gesetzt ist. Interessant ist dabei der Fall, wenn beispielsweise bei angeschlossenem Motor A der Pegelwert der beiden Kanäle 0 und 1 unterschiedlich ist. Schaltet man den Motor mit den Pfeiltasten abwechselnd zwischen Links- und Rechtslauf um, dann dreht er sich in die verschiedenen Richtungen auch mit unterschiedlicher Geschwindigkeit. In der Wirklichkeit kommt dies z. B. bei einer Werkzeugmaschine vor, wo die Vorschubgeschwindigkeit beim Arbeitsgang im Normalfall geringer ist als beim Zurückfahren in die Ausgangsstellung.

Bei der Direktsteuerung von LEGO Modellen läßt sich diese unterschiedliche Pegeleinstellung für die beiden Drehrichtungen eines Motors nutzen beim Steuern des Robotermodells. Befindet sich Transportgut im Greifer, dann wird die Drehung des Roboterarms vom Aufnahmeort zum Ablageort langsamer und behutsamer durchgeführt als die Rückbewegung des Roboterarms mit leerem Greifer.

Wie wird nun beim LEGO Interface diese unterschiedliche Leistungseinstellung der Ausgänge technisch realisiert? Man könnte dies durch ein Absenken der an den Ausgängen anliegenden Spannung bewerkstelligen. Die Motoren und Lampen würden dann mit weniger Strom versorgt werden. Diese Methode wäre jedoch kompliziert und teuer, weil dann auch im Interface weitere komplexe elektronische Bauteile nötig wären.

Viel einfacher ist es, die unterschiedlichen Leistungspegel softwaremäßig zu realisieren. Man läßt dazu die volle gleichbleibende Spannung immer wieder automatisch vom Computer für einen kurzen Augenblick wegschalten. Über die Zeit gemittelt bekommen dann die angeschlossenen Geräte auch weniger Strom als bei dauernd anliegender Spannung.

Um einen bestimmten Leistungspegel im Vergleich zum maximalen zu erreichen, muß man die Spannung für eine bestimmte Zeitspanne im Verhältnis zur Einschaltdauer wegnehmen. Dies kann man in kürzeren oder längeren Zeitabständen tun. Die Zeitdauer einer Ausschaltphase darf dabei aber nur nicht so lang werden, daß der Motor zwischenzeitlich stehen bleibt oder eine angeschlossene Lampe ausgeht.

Uns hat die Dauer dieser Ausschaltintervalle genauer interessiert, und darum haben wir sie mit einem Oszilloskop sichtbar gemacht und die Impulszeiten über die Ablenkstrecken des Elektronenstrahls gemessen. Die gemittelten Meßwerte für die Schaltimpulse bei den einstellbaren Leistungspegeln sind in der nachfolgenden Tabelle (Abb. 7.5) zusammengestellt.

Die Meßergebnisse zeigen, daß sowohl das Ein- wie das Ausschaltintervall über die Leistungsstufen hinweg geändert wird. Mit diesem gegenläufigen Zeitverhalten erreicht man, daß insbesondere die Ausschaltintervalle nicht zu lang werden und den Steuerungsablauf beispielsweise bei irgendwelchen Zählvorgängen beeinflussen können.

Leistungs-pegel	Einschaltintervall in Millisekunden	Ausschaltintervall in Millisekunden
8	-	0
7	6,3	0,5
6	2,8	0,6
5	2,0	0,6
4	1,1	0,6
3	1,1	1,5
2	1,1	2,3
1	1,1	6,6
0	0	-

Abb. 7.5 Meßtabelle der Leistungspegelimpulse

Eine physikalische Betrachtungsweise der Meßtabelle zeigt unter Berücksichtigung der Genauigkeit der gemittelten Meßwerte, daß die Autoren des TC Controllers sich bemüht haben, die elektrische Leistung linear herunterzufahren. Die verfügbare Leistung läßt sich nämlich über die Beziehung

$$\text{el. Leistung} \sim [\text{Einschaltzeit}/(\text{Einschaltzeit}+\text{Ausschaltzeit})]^2$$

ermitteln. Die nach dieser Beziehung ausgewerteten Messungen ergeben das nachfolgende Säulendiagramm (Abb. 7.6) mit den bei den einzelnen Leistungspegeln verfügbaren Leistungen, welche jeweils auf die maximal mögliche Leistung bezogen sind.

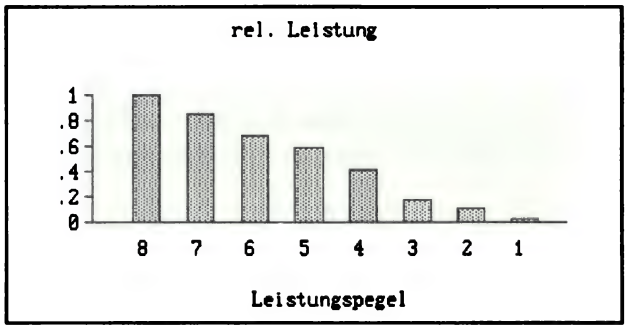


Abb. 7.6 Relative Leistung der verschiedenen Leistungspegel

7.2.5 Regelprozesse mit der Direktsteuerung

Nachdem wir uns bislang wesentlich nur mit den Ausgabekanälen des Interfaces beschäftigt haben, wollen wir nun auch die Eingabekanäle bei der Direktsteuerung genauer betrachten. Über Eingangskanäle bekommt die Steuerungszentrale die Informationen, die für den weiteren Ablauf des zu steuernden Prozesses eventuell nötig sind. Von reinen Zeitsteuerungen abgesehen ist nämlich bereits bei einfachen Steuerungsprozessen, wie beispielsweise einer bedarfgesteuerten Fußgängerampel, die dauernde Kontrolle eines Eingabesignals erforderlich, um prozeßgerecht schalten und reagieren zu können. Regelprozesse erfordern dann noch eine viel differenziertere Abfrage des augenblicklichen Prozeßzustandes.

Mit dem LEGO Interface haben wir zwei digitale Eingabekanäle zur Verfügung, die mit den Portnummern 6 und 7 bezeichnet sind. Digital heißt hier, daß nur zwei Signalzustände existieren, die auch im Fenster der Direktsteuerung angezeigt werden. Um den Zustand der Eingabeleitungen untersuchen zu können, müssen wir aber erst noch geeignete Sensoren, wie den Tastensensor oder den Optosensor, anschließen. Denn nur über externe Signale läßt sich der Zustand dieser Leitungen beeinflussen. Leuchtet die grüne Anzeigelampe des Eingabekanals, wird im Fenster der Wert 1 angezeigt. Bei dunkler Anzeigelampe ist der Wert des Signalzustandes 0. Beim Manipulieren am Sensorbaustein wechselt dieser digitale Zustand entsprechend. Ist ein Eingabekanal nicht mit einem Sensorbaustein beschaltet, hat er in jedem Fall den Signalwert 0.

Bei der Direktsteuerung eines Modells wie der bedarfgesteuerten Fußgängerampel muß der Operateur am Schaltpult, an der Tastatur, die Anzeigelampe oder das Bildschirmfenster der Direktsteuerung ständig im Auge haben, wenn er auf das Anfordern einer Fußgängerphase bedarfs- und situationsgerecht reagieren will. In einem Programm kann diese Überwachungsfunktion der Computer übernehmen.

7.2.6 Zähler bei der Direktsteuerung

Nun ist neben der Zustandsanzeige noch die Anzeige für einen Zähler A und einen Zähler B im Direktsteuerungsfenster vorhanden. Zähler A ist optisch dem Eingabekanal 6 und Zähler B dem Eingabekanal 7 zugehörig. Was zählen diese Zähler?

Man ist versucht anzunehmen, daß mit diesen Zählern Signalzustandswechsel auf den Eingabeleitungen 6 und 7 gezählt werden und liegt mit dieser Annahme auch nicht falsch, wie wir später sehen werden. Nur wenn man unmittelbar nach der Installation des TC Controllers diese Zähler ansprechen möchte, indem man mehrmals den Tastensensor betätigt oder das Fenster des Optosensors freigibt - und dabei den Zustandswechsel auch auf dem Bildschirm beobachten kann -, ist man vom Ausbleiben jeder Zählerreaktion enttäuscht. Wie lange man auch mit den Sensoren spielt, der Zählerstand bleibt auf der Anzeige 0.

Vielleicht kann das Hilfefenster weiterhelfen, welches wir uns wieder mit **<F10>** holen. Dort bekommen wir einen Hinweis, wie wir die Zähler stellen können: Mit einer Tastenkombination - entweder linke oder rechte **<Shift>**-Taste und **<6>** bzw. **<7>** - kann man jeden der beiden Zähler über Tastendruck vorwärts und rückwärts zählen lassen. Ein kurzes Antippen der Tasten **<6>** bzw. **<7>** setzt die Zähler dann wieder auf den Wert 0 zurück. Dieses Einstellen und Rückstellen der Zähler über die Tastatur kann aber doch bestimmt nicht deren alleinige Daseinsberechtigung sein.

Wir schließen darum in unserem Erkundungsdrang im nächsten Schritt an den Ausgangskanal 0 einen Lampenbaustein und schalten die Lampe mit der Taste **<0>** ein und aus. Das Licht detektieren wir mit einem Optosensor. Beim Zähler im Fenster der Direktsteuerung zeigt sich aber immer noch nichts. Oder hat die Bezeichnung der Zähler mit A und B mit den gleichbenannten Motorsteuerungskanälen A und B etwas zu tun?

Wir schalten darum nun die am Ausgangskanal 0 angeschlossene Lampe mit der **<→>**-Taste. Und siehe, es tut sich was am Zähler A. Hätten wir das Softwarehandbuch bereits am Anfang unseres Erforschens zur Hand genommen, hätten wir eventuell gleich unser Erfolgserlebnis haben können. Dort heißt es nämlich: "Die Direktsteuerung stellt als Meßhilfe zwei Zähler A und B bereit, die mit den Motoren an Ausgang A und B gekoppelt sind. Der Zähler A wertet die Impulse am Eingang 6 aus; jeder Übergang von 'ein' zu 'aus' oder 'aus' zu 'ein' der grünen Eingangskontrollampe 6 verändert den Zählerstand um eine Einheit. ..."

Beim weiteren Spielen mit diesen beiden Zählern erhellt sich uns nun die Handbuchformulierung ".. die mit den Motoren A und B gekoppelt sind". Die Kopplung besteht darin, daß mit einem bereits einmaligen

Antippen der Motorsteuerungstasten <→> und <←>, bzw. <↑> und <↓> der Zähler A, bzw. B 'scharf' gemacht wird und dann auf jeden Signalzustandswechsel am Kanal 6, bzw. 7 reagiert und den Wechsel zählt. Wo und wie dieser Signalwechsel zustande kommt, spielt nun keine Rolle mehr. Der Zähler ist nach dem Einschalten in ständiger Bereitschaft. In dieser Bereitschaftsposition kann jedoch die Zählrichtung durch das Antippen der jeweils anderen Pfeiltaste beliebig oft umgestellt werden.

Innerhalb der Direktsteuerung ist es nicht möglich, die Zählbereitschaft wieder zurückzunehmen. Man muß den TC Controller dazu schon aus dem Speicher entfernen und neu laden. Das Entfernen des TC Controller-Programms aus dem Speicher gelingt ohne erneutes Booten des Betriebssystems, wenn man die Direktsteuerung nicht mit **<Esc>** sondern mit der Tastenkombination **<Alt>-<x>** wegblendet. Voraussetzung dafür ist allerdings, daß nach dem Laden des TC Controllers kein weiteres speicherresidentes Programm mehr nachgeladen wurde.

Die Zähler in der Direktsteuerung sind eine wesentliche Neuerung in dem Steuerungskonzept von LEGO. Mit einfachen Lichtschranken läßt sich bei unterschiedlichen Zählaufgaben der augenblickliche Zählerstand direkt auf dem Bildschirm verfolgen. Besonders vorteilhaft kann dies z. B. für das genaue, reproduzierbare Anfahren irgendwelcher Positionen bei der Handsteuerung des LEGO Robotermodells oder der im Projekt 4.4 behandelten Bahnschranke sein.

Die bislang noch nicht angesprochene letzte Zeile im Direktsteuerungsfenster baut uns schließlich eine Brücke in den zweiten Teil des LEGO TC Controllers. Dort sind nämlich diejenigen Pascal-Befehle eingeblendet, die der gerade initiierten Aktion innerhalb der Direktsteuerung entsprechen. Dadurch läßt sich, wie wir später zeigen werden, eine Aufgabenstellung, die sich auch mit den Möglichkeiten der Direktsteuerung erledigen läßt, praktisch unmittelbar in ein Steuerungsprogramm in der Programmiersprache Pascal umsetzen. Die Direktsteuerung ist so eine anschauliche Hilfe für den Entwurf eines Pascal-Programms.

Bevor wir an eine in Turbo Pascal programmierte Steuerung herangehen können, müssen wir noch ein paar Vorarbeiten leisten. Die speziellen, im Direktsteuerungsfenster eingeblendeten Steuerungsprozeduren des LEGO TC Controllers müssen nämlich auch dem Pascal-Compiler bekannt gemacht werden.

7.3 Der LEGO TC Controller als Unit in Turbo Pascal

Um den Standard-Sprachumfang von Pascal auf spezifische Aufgabenstellungen hin zu erweitern, verwendet Turbo Pascal das Konzept der Units. In einem Unit - das grammatikalische Genus im Turbo Pascal Handbuch dafür ist das Neutrum: das Unit - werden die jeweiligen problemorientierten Prozeduren und Funktionen zusammengefaßt.

Die LEGO TC Controller-Prozeduren sind in dem Unit LTCC.TPU enthalten, die sich als Objektcode auf der Master Disk des LEGO TC Controllers befindet. Jede Objektcodedatei wurde von dem Quellcodeprogramm durch Compilieren, d. h. Übersetzen in eine für den Prozessor verständliche Binärdatei erhalten. Sie läßt dadurch naturgemäß die Eigenheiten des Compilers erkennen und ist daher auch nur mit der zugehörigen Compilerversion zu verwenden.

Für Turbo Pascal sind in den letzten Jahren neue Versionen, sog. Upgrades, erschienen. Daher finden sich auch auf der Master Disk des LEGO TC Controllers mehrere Objektcodeversionen des Units LTCC.TPU. Das LTCC-Unit gibt es zur Zeit von der Turbo Pascal Version 4.0 bis zur Version 5.5 und für Quick Pascal 1.0. Für die verschiedenen Versionen befinden sich diese Units mit den Steuerungsprozeduren jeweils in eigenen Unterverzeichnissen auf der Master Diskette bzw. der Update Diskette zum LEGO TC Controller.

Wir verwenden Turbo Pascal in der Version 5.5 (ohne jedoch in unseren Beispielen die neuen Möglichkeiten der objektorientierten Programmierung zu nutzen). Die Unit-Dateien sind dazu im Unterverzeichnis LTCCTP55 der Master Disk (Update Diskette zum LEGO TC Controller). Von den beiden Dateien ist LTCC2.TPU das entsprechende Unit für den möglichen Anschluß und die Steuerung eines zweiten LEGO Interfaces an einem Computer. Wir beschränken uns bei der Vorstellung des LEGO TC Controllers jedoch auf ein Interface.

Da das Steuerungsunit LTCC.TPU von uns nun dauernd verwendet wird, ist es sinnvoll, die darin definierten Prozeduren und Funktionen in die .TPL-Datei (Turbo Pascal Library) aufzunehmen, weil es dann automatisch im Hauptspeicher des Computers vorliegt und nicht bei jedem Bedarf nachgeladen werden muß. Dies geschieht mit dem Programm TPUMOVER von Turbo Pascal.

Beim Erstellen der Arbeitsdiskette für die Direktsteuerung haben wir schon auf das Problem aufmerksam gemacht, vor dem wir nun stehen. Ab der DOS-Version 3.3 reicht neben den für das Booten des Betriebssystems unbedingt notwendigen Dateien und dem LEGO TC Controller-Programm CONTROL.EXE auf einer 5,25" Diskette mit 360 KByte der freie Platz nicht mehr aus, um auch noch das Turbo Pascal-Compilerprogramm TURBO.EXE und die Turbo Pascal-Library TURBO.TPL aufzunehmen.

Wir richten uns daher zweckmäßigerweise noch eine zweite Arbeitsdiskette her, auf die wir die Turbo Pascal-Dateien bringen, die für das Schreiben und Testen der Steuerungsprogramme für den LEGO TC Controller notwendig sind. Auf dieser Diskette können wir dann auch die von uns geschriebenen Steuerungsprogramme abspeichern.

Auf eine als Datendiskette formatierte Diskette in Laufwerk A: kopieren wir fürs erste von der Master Disk in Laufwerk B: mit

```
copy b:\ltcctp55\ltcc.tpu a:
```

das Steuerungsunit LTCC.TPU. Dann tauschen wir die Diskette in B: gegen die Disketten mit den Turbo Pascal-Dateien und übertragen von dort nach A: die Dateien

```
TURBO.EXE
TURBO.TPL
TPUMOVER.EXE
```

Nun starten wir vom aktuellen Laufwerk A: aus das Programm TPUMOVER. Es erscheint auf dem Bildschirm der aktuelle Bestand der Turbo Pascal Library (Abb. 7.7). Von dem linken Rahmen schalten wir, wie in der Befehlszeile angezeigt, mit **<F6>** auf den rechten Rahmen um und laden mit **<F3>** und zweimaligem **<Return>** neue Units in den Rahmen. In unserem Fall ist dies allein das Unit LTCC.TPU. Nach dem Markieren dieses Unit mit **< + >** fügen wir es dann durch Drücken der Einfügetaste **<Ins>** neu in die Unit-Liste des linken Rahmens (Abb. 7.7). Nun müssen wir die Taste **<Esc>** drücken, um die Frage vorgelegt zu bekommen, ob wir diese Änderung abspeichern wollen: die Frage beantworten wir mit **<y>**es. Nun ist das Unit LTCC.TPU in die Prozedurbibliothek TURBO.TPL aufgenommen.

Die beiden Dateien LTCC.TPU und TPUMOVER.EXE auf der Diskette haben ihren Dienst getan und können nun schadlos gelöscht werden. Leider bekommen wir auch dadurch nicht den freien Platz, um noch wenigstens die kürzere englische Version der Datei

TURBO.HLP

auf unsere zweite Arbeitsdiskette laden zu können. Bei der Erstellung unserer Steuerungsprogramme können wir daher in der integrierten Entwicklungsumgebung von Turbo Pascal nicht über die Funktionstaste **<F1>** auf die mögliche kontextbezogene Hilfe zurückgreifen - wenn wir nicht extra noch zusätzlich eine dritte Diskette mit dieser Datei bereithalten und sie jedesmal vor dem Drücken von **<F1>** ins Laufwerk schieben.

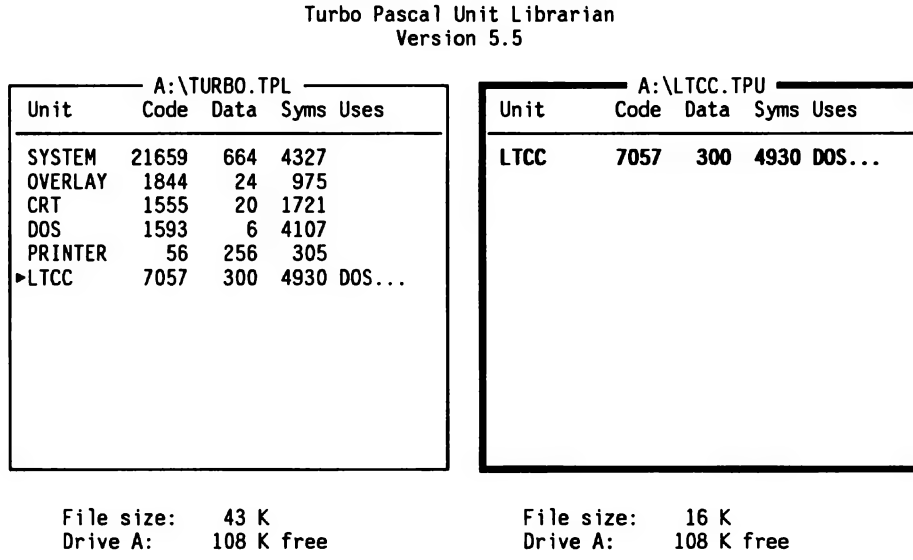


Abb. 7.7 Einbindung des Unit LTCC.TPU in die Bibliothek TURBO.TPL

Es fällt uns daher nicht schwer, den Komfort zu werten und zu schätzen, auf den man beim Arbeiten mit einer Festplatte zurückgreifen kann. Ebenso werden andererseits auch die Vorzüge von neuen Computerkon-

figurationen deutlich, die ein Laufwerk für Disketten mit einer größeren Speicherkapazität als bootfähiges Laufwerk A: besitzen. Denn schon auf einer 720 KByte Arbeitsdiskette lassen sich alle unsere Dateien einschließlich der Hilfedatei von Turbo Pascal speichern.

Nach diesen Vorarbeiten wollen wir aber endlich zum Erkunden des Teils des LEGO TC Controllers in der Turbo Pascal Umgebung kommen. Wir booten dazu mit der ersten Arbeitsdiskette unser System und den TC Controller. Jetzt ist dieser mit seinen Direktsteuerungsmöglichkeiten verfügbar. Von der zweiten Arbeitsdiskette laden wir von der Betriebssystemebene mit

```
A> turbo
```

den Turbo Pascal-Compiler hinzu (Abb. 7.8). Die Prozedurbibliothek TURBO.TPL wird vom Compiler geladen und genutzt, wenn wir ein Steuerungsprogramm geschrieben haben und dieses in Binärcode umsetzen lassen oder starten.

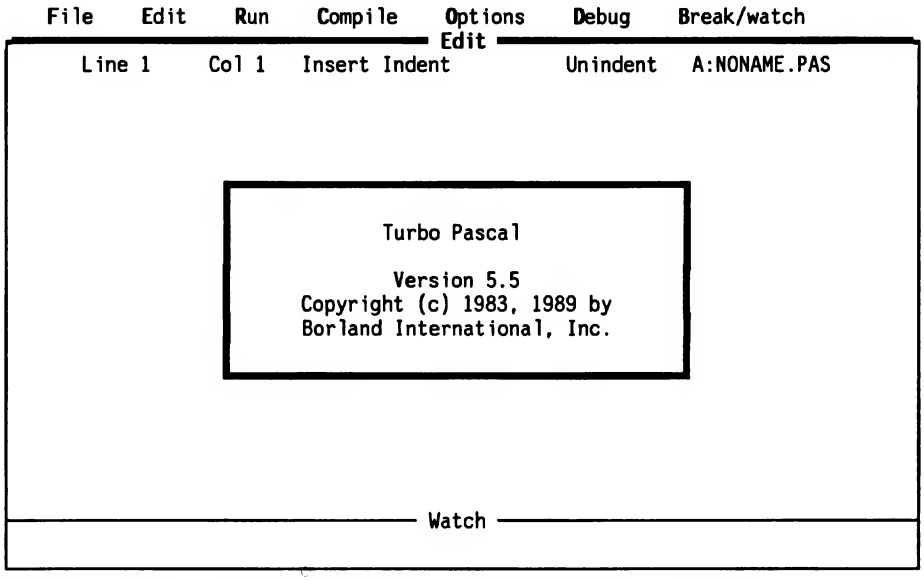


Abb. 7.8 Die integrierte Entwicklungsumgebung von Turbo Pascal

7.4 Die ersten Steuerungsprogramme in Turbo Pascal

Nach dem Laden von Turbo Pascal haben wir den Bildschirm der integrierten Entwicklungsumgebung vor uns (Abb. 7.8). Das Zurechtfinden in ihr soll im Folgenden nur insoweit kurz beschrieben werden, wie es für unsere ersten Arbeiten notwendig ist.

Mit den Pfeiltasten <→> bzw. <←> bewegen wir die Markierung in der Menüzeile auf den Menüpunkt Edit und bringen dann mit **<Return>** den Cursor in das Editierfeld. Eilige können dieses Ergebnis des mehrmaligen Tastendrückens auch gleich mit dem Drücken der Tastenkombination **<Alt>-<e>** erreichen.

Wir wollen nun in Pascal unseren ersten Steuerungsbefehl schreiben und dabei das Signal auf dem Ausgabeportbit 0 des LEGO Interface auf 1 setzen, so daß die Kontrollampe dort aufleuchtet. Erinnern Sie sich noch an den Befehl, der in der untersten Zeile des Rahmens der Direktsteuerung eingeblendet war, als wir dies in der Direktsteuerung machten? Wenn nicht, ist es nicht weiter schlimm. Der TC Controller wartet mit seiner Direktsteuerung ja förmlich im Hintergrund darauf, daß wir ihn herholen. **<Alt>-<i>**, und schon sind wir wieder in der uns bereits bekannten Umgebung.

Beim Antippen der Taste **<0>** erscheint im Rahmen

letzte Anweisung: BitOn(0)

bzw. beim Zurücksetzen des Signals das Befehlswort

BitOff(0)

Die Zahl in der Klammer des Befehls bezeichnet die jeweilige Ausgabeleitung.

Mit **<Esc>** schalten wir die Direktsteuerung ab und schreiben nun unser Programm. Denken wir noch daran, daß am Anfang eines Programms oder eines Programmblocks in Pascal das Befehlswort **begin** steht und das Programm mit **end** abgeschlossen werden muß, so formulieren wir wahrscheinlich unser Programm folgendermaßen:

```
begin
  BitOn(0)
end.
```

Der Punkt im Programm nach **end** darf nicht vergessen werden. Er zeigt an, daß das Hauptprogramm hiermit abgeschlossen ist.

Versuchen wir nun, dieses Programm zu compilieren oder zu starten, so bekommen wir im Editierfeld die Fehlermeldung

Error 3: Unknown identifier

eingeblendet und der Cursor steht beim Befehlswort **BitOn**(0). Was ist in unserem Programm noch falsch?

Die Programmiersprache Pascal ist bekannt dafür, daß jedem Anweisungsteil eines Programms ein Deklarationsteil voranstehen kann oder sogar vorangehen muß, wenn im Anweisungsteil irgendwelche Variablen, Prozeduren u.a. verwendet werden, die anderweitig noch nicht deklariert oder für diesen Programmteil nicht gültig sind.

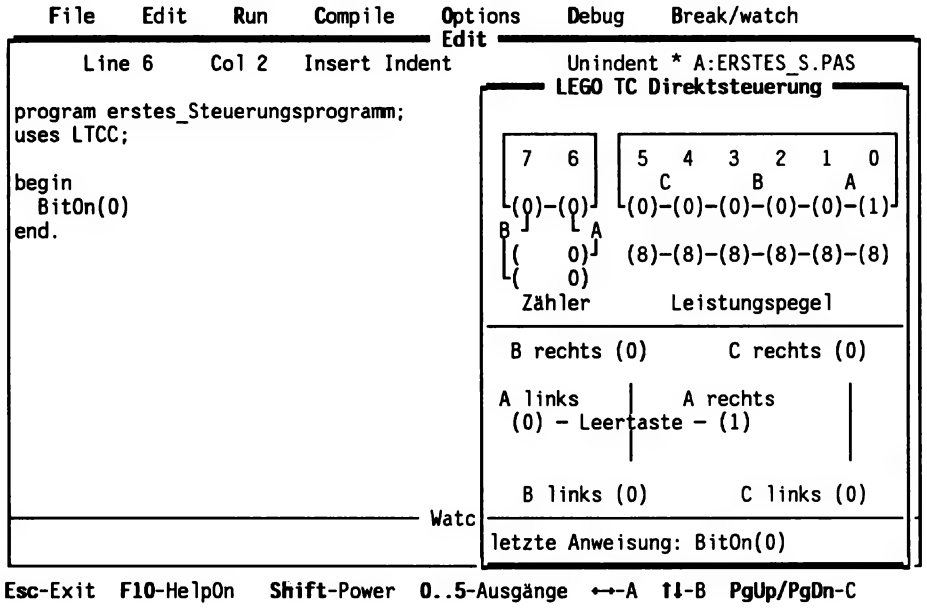


Abb. 7.9 erstes_Steuerungsprogramm in Turbo Pascal mit dem TC Controller

Die Prozedur **BitOn(0)** ist keine Standardprozedur von Pascal und muß deklariert werden. Sie wurde von uns nur mit dem Unit LTCC.TPU in die Pascal-Bibliothek .TPL aufgenommen. Es muß dem Compiler mit

uses LTCC;

bekannt gemacht werden, daß er sie dort vorfindet. Mit dem reservierten Turbo Pascal-Begriff **uses** werden dem Compiler all diejenigen Units mitgeteilt, aus denen anwender- und problemspezifische Prozeduren, Funktionen und Datendeklarationen im nachfolgenden Programm genommen sind. Die in der Bibliothek verfügbaren Units können wir aus der Abb. 7.7 ersehen. Diese zeigt einen Arbeitsbildschirm bei der Einstellung des Units LTCC.TPU in die Bibliothek.

Der Strichpunkt hinter der Deklaration trennt wie in Pascal üblich und notwendig einzelne Deklarations- und Programmstatements. Bei unserem dreizeiligen Programm wäre ein Strichpunkt nach **BitOn**(0); ebenfalls möglich; er ist jedoch hier nicht unbedingt nötig, da gleichsam das nachfolgende **end** seine Separatorfunktion übernimmt.

Einem angemessenen Gebot nach Transparenz und Ordnung beim Programmieren folgend ist es zudem empfehlenswert, den Programmen nicht nur zum Abspeichern einen Namen zu geben. Dies erfolgt in Turbo Pascal mit dem reservierten Wort **program** am Programmanfang als Teil der Deklaration. Unser **erstes_Steuerungsprogramm** ist damit vollständig und lauffähig. Es ist in der Bildschirmkopie von Abb. 7.9 dargestellt.

Bei der Deklaration und Definition von Namen, Variablen und anderen sog. Bezeichnern sind bei Turbo Pascal verschiedene Regeln und Einschränkungen zu beachten. Die wichtigsten sind:

- Bezeichner müssen mit einem Buchstaben beginnen (A .. Z, a .. z).
- Nach dem ersten Buchstaben können auch Ziffern und Unterstriche folgen; Umlaute, ß und Leerzeichen sind nicht erlaubt.
- Zwischen Groß- und Kleinschreibung wird nicht unterschieden.

Die Compilierung unseres Programms mit **<Alt>-<F9>** zeigt uns, daß dieses nun fehlerfrei ist. Nach dem Programmstart mit **<Ctrl>-<F9>** sehen wir, daß das Programm auf dem Interface auch die Kontrollampe am Ausgang 0 einschaltet, so wie wir es wollten.

Von unserem ersten Programmiererfolg mit dem LEGO TC Controller in Turbo Pascal angespornt, versuchen wir uns gleich an Lauflichtprogrammen, mit denen wir auch bei LEGO Lines gestartet sind.

```
program Lauflicht_0;
uses LTCC;
begin
  BitOn(0);
  BitOff(0);
  BitOn(1);
  BitOff(1);
  BitOn(2);
  BitOff(2);
  BitOn(3);
  BitOff(3);
  BitOn(4);
  BitOff(4);
  BitOn(5);
  BitOff(5);
end.
```

Bei dem Erstellen und Erproben des Programms **Lauflicht_0** fiel uns, wenn wir uns an LEGO Lines erinnern, vielleicht auf:

- Jede eingeschaltete Lampe muß mit einem Befehl **BitOff**(Port) auch wieder ausgeschaltet werden.
- Das Turbo Pascal Steuerungsprogramm wird so viel schneller abgearbeitet als ein gleiches Programm in LEGO Lines, daß wir beim obigen Lauflicht_0 das Aufleuchten der Lampen gar nicht mehr wahrnehmen.

Wenn wir die Einschaltzeit der einzelnen Lampe verlängern wollen bzw. müssen, dann können wir in unserem Programm zwei Wege beschreiten:

1. Wir programmieren nach jedem Einschaltbefehl beispielsweise eine eigene Programmsequenz zur Verlangsamung ein oder
2. wir verwenden eine bereits in dem LTCC-Unit vorhandene Warteprozedur. Im nächsten Abschnitt lernen wir die Prozedur **Wait**(Zeit) kennen, die den Programmfortgang um die angegebene Zeit anhält.

Wir entscheiden uns hier für den zweiten Fall. Außerdem soll das Lauflichtprogramm nicht nur einmal abgearbeitet werden, sondern erst durch einen Tastendruck beendet werden. Dazu können wir die Schleifenstruktur

```
repeat
   Anweisungsblock
until (Bedingung)
```

von Pascal verwenden. Als Bedingung nehmen wir die in dem Unit Crt definierte **KeyPressed**-Funktion. KeyPressed ergibt den Wert "wahr", wenn eine Taste gedrückt wurde.

Schließlich wollen wir uns bei dem Lauflichtprogramm mit seinen 6 Lampen auch noch die Möglichkeit von Zuweisungen über Variablen zunutze machen. Wir müssen nicht für jede Lampe einen eigenen Befehl ins Programm schreiben, wenn wir das Ein- und Ausschalten der einzelnen Lampen über die Laufvariable *Port* besorgen. Diese Variable muß natürlich in Pascal einschließlich ihres Typs deklariert werden. Für die Schleifenstruktur der Laufvariablen wählen wir die **for**-Schleife.

Das endgültige Programm lautet damit folgendermaßen:

```
program Lauflicht_1;
uses LTCC, Crt;
var
    Port  : Byte;
begin
    repeat
        for Port := 0  to 5 do
            begin
                BitOn (Port);
                Wait (0.5);
                BitOff (Port);
            end;
    until KeyPressed;
end.
```

Nach jedem Durchgang, bei dem die einzelnen Lampen der Reihe nach ein- und dann wieder ausgeschaltet werden und der insgesamt etwa 3 Sekunden dauert, wird jedesmal vom Programm abgefragt, ob durch einen zwischenzeitlichen Tastendruck das Programmende signalisiert wurde. Die Programmierung ist gegenüber LEGO Lines interessanter und flexibler geworden, da wir neben den Prozeduren und Funktionen des TC Controller spezifischen Units LTCC natürlich auch die ganzen Möglich-

keiten der mächtigen Hochsprache Turbo Pascal zum Einsatz bringen können.

Es ist nicht Ziel dieses Buches oder dieses Kapitels, eine Einführung in die Grundlagen über die Programmierung mit Turbo Pascal oder gar einen Aufbaukurs darüber zu geben. Hier verweisen wir Sie auf andere Literatur, z.B. das Buch 'PC-Nutzung mit Turbo Pascal' von Klaus Horn. Wir müssen nach dieser kurzen Einführung, die nur den eventuellen Anfängern die ersten und elementaren Schritte in der Programmierung mit Turbo Pascal aufzeigen sollten, die Kenntnis der Programmiersprache Pascal und dieser Programmierumgebung im Wesentlichen voraussetzen. Wir können und wollen mit unserer knappen Darstellung der Programmierung mit dem TC Controller die Leistungsfähigkeit dieses Instruments bei der Lösung von Steuerungs- und Regelungsproblemen nur anreißen und der Lehrerschaft damit eine Orientierungs- und Entscheidungshilfe geben, wenn es darum geht, mit welchen Medien unterrichtliche Aufgabenstellungen angegangen werden sollen. Nach der nachfolgenden lexikalischen Darstellung aller durch das Unit LTCC verfügbaren neuen Prozeduren und Funktionen in Turbo Pascal, werden wir zum Ende des Buches dann noch zwei Beispielprogramme bringen.

7.5 Die Prozeduren und Funktionen des LEGO TC Controllers

Einer Reihe der in dem Unit LTCC.TPU für Turbo Pascal definierten Prozeduren und Funktionen haben wir schon bei der Direktsteuerung begegnen und ihre Wirkung erfahren können. Bei der anschließenden vollständigen Darstellung der auf die spezifische Hardwareumgebung zugeschnittenen und dem Anwender verfügbaren Prozeduren und Funktionen des Units LTCC.TPU orientieren wir uns zwar weitgehend am Softwarehandbuch zum LEGO TC Controllers, es fließen aber über das Handbuch hinaus in die Erläuterungen auch eigene Erfahrungen mit ein. Mit der Tastenkombination **<Alt>-<h>** können Sie sich übrigens bei geladenem Controller jederzeit die verschiedenen Routinen mit Kurzinformationen anzeigen lassen.

Bei eventuellen Programm- und Anweisungsbeispielen ist die in jedem Fall nötige Deklaration **uses LTCC** fast immer weglassen. Ähnlich sind wir auch mit den Programmblockwörtern **begin** und **end** verfahren.

Bezüglich der vollen Programmstruktur sei auf die Abb. 7.9 oder das Lauflichtprogramm von Seite 174 verwiesen.

7.5.1 Steuerung der Interface-Ausgänge

BitOn-Prozedur

Funktion: schaltet den Ausgang Port ein.

Deklaration: BitOn (*Port* : Byte)

Erläuterung: Für den Übergabeparameter *Port* sind die Werte aus dem Bereich [0 .. 5] gültig. Andere Werte erzeugen beim Programmlauf die Fehlermeldung wie beispielsweise "Der Anschluß 7 ist ein Eingang!" oder "Der Anschluß 56 existiert nicht!". Beim Compilieren wird ein falscher Parameterwert außerhalb des gültigen Wertebereichs noch nicht erkannt.
Wird *Port* beim Prozeduraufruf mit einer Variablen übergeben, so muß diese vorher wie in unserem obigen Programmbeispiel als Datentyp Byte deklariert sein.

Bei Programmende werden die gesetzten Ausgänge nicht automatisch zurückgesetzt.

Querverweis: BitOff, GetBit, Motor, SetPower, FlashOff

BitOff-Prozedur

Funktion: schaltet den Ausgang Port aus.

Deklaration: BitOff (*Port* : Byte)

Erläuterung: Vergleichen Sie die Erläuterung zur BitOn-Prozedur.

Querverweis: BitOn, GetBit, Motor, SetPower, FlashOff

Motor-Prozedur

Funktion: schaltet Paare von Ausgängen zur Richtungssteuerung von Motoren.

Deklaration: Motor (*Port* : Byte, *M* : Mode)

Erläuterung: Für den Parameter *Port* sind die Buchstaben A, B, C oder auch a, b und c gültige Werte.

Der Parameter *M* ist als Typ Mode definiert. *M* kann einen Wert aus Mode = (links, rechts, wechseln, halten, lft, rgt, trn, hlt) annehmen.

Die Kurzbezeichnungen sind von den entsprechenden englischen Wörtern abgeleitet. Der Wert 'wechseln' schaltet die Laufrichtung des Motors unabhängig von seinem gerade aktuellen Zustand in die andere Drehrichtung; mit 'halten' wird das Portpaar auf den Signalzustand 0 gesetzt.

Übergabeparameter außer den aufgelisteten führen bereits bei der Programmcompilierung zu einer Fehlermeldung.

Da im Unit-Programm die Werte von *Port* in die Zahlenwerte 21, 22 und 23 umgewandelt werden, sind jedoch auch gleich diese Werte als Übergabeparameter ohne Fehlermeldung möglich.

Querverweis: BitOn, BitOff, GetBit, SetPower

Beispiel: Motor(a,links)

7.5.2 Steuerung der Ausgangsleistung

SetPower-Prozedur

Funktion: steuert die Ausgangsleistung *Level* des Ausgangs *Port* durch die schnelle Folge von Spannungsimpulsen.

Deklaration: SetPower (*Port* , *Level* : Byte)

Erläuterung: Der Parameter *Port* kann einen Wert aus [0 .. 5] oder aus [A, B, C, a, b, c] annehmen. Für *Level* sind die Werte [0 .. 8] gültig.

Mit dieser Prozedur werden an den Ausgängen die Leistungspegel von 0 bis 8 eingestellt. Der Wert 8 entspricht der vollen Leistung und der Wert 0 schaltet die spannung am Ausgang dauernd ab. Bei den Pegelwerten 1 bis 7 wird die Spannung in unterschiedlichen Zeitintervallen zu- und abgeschaltet (vgl. Abb. 7.5).

Die Änderung des Leistungspegels im Programm kann auch bei laufendem Motor erfolgen. Mit Portparametern [0 .. 5] lassen sich an den beiden Ports eines Motorausgangs unterschiedliche Leistungspegel einstellen.

Querverweis: GetPower, SetFlash

Beispiel:
```
SetPower(B,8);
Motor(B,links);
Wait(5);
SetPower(B,5);
Wait(5);
```

SetFlash-Prozedur

Funktion: schaltet den Ausgang *Port* für die Zeit *Mark* ein und die Zeit *Space* aus.

Deklaration: SetFlash (*Port* : Byte; *Mark*, *Space* : Real)

Erläuterung: Für den Parameter *Port* sind Werte aus [0 .. 5] gültig, für *Mark* und *Space* Werte von 0.1 bis 25.5.

Die Prozedur gestattet für die verschiedenen Ports je ein eigenes Puls-Pausenverhältnis zu programmieren. Gegenüber der fest vorgegebenen Pegeleinstellung mit SetPower sind mit SetFlash längere Takt-/Schaltzeiten möglich. Der Leistungspegel mit SetPower kann zusätzlich aktiviert werden.

Die Steuerzeiten für 'ein' bzw. *Mark* und 'aus' bzw. *Space* können mit einer Auflösung von 0,1 Sekunden angegeben werden können. Die Voreinstellung der Flash-Prozedur ist 0.5, was einen Gleichtakt von je 0,5 Sekunden ergibt.

Das mit SetFlash eingestellte Taktsignal wird erst mit der Prozedur FlashOn gestartet. Es kann auch im laufenden Betrieb neu gesetzt werden.

Querverweis: FlashOn, FlashOff, SetPower

Beispiel:
```
SetFlash(4,1.5,2);
FlashOn(4);
{Signallampe von Port 4 blinkt: 1,5 s ein und 2 s aus}
SetFlash(4,1.2,0.7);
{Signallampe von Port 4 blinkt: 1,2 s ein und 0,7 s aus}
```

FlashOn-Prozedur

Funktion: schaltet die mit SetFlash eingestellten Pulszeiten am Ausgang *Port* ein.

Deklaration: FlashOn (*Port* : Byte)

Erläuterung: Der Parameter *Port* kann die Werte [0 .. 5] annehmen.

Die Prozedur schaltet an dem bezeichneten Steuerausgang ein programmiertes Taktsignal ein. Die Taktzeiten betragen je 0,5 Sekunden für 'ein' und 'aus', wenn sie nicht mit SetFlash anders gesetzt worden sind.

Eine im Programm nachfolgende BitOff oder BitOn-Anweisung setzt das Blinken außer Kraft. Anders verhält sich eine Motor-Anweisung wie beispielsweise Motor (A,rechts), die einen vorangehenden FlashOn(0) Befehl unbeschadet weiter aktiv sein läßt: der Motor geht an und aus.

Querverweis: SetFlash, FlashOff, SetPower, BitOff, BitOn

Beispiel:
```
SetFlash(3,7.5,2.5);
FlashOn(3);
Motor(B,links);
```

FlashOff-Prozedur

Funktion: schaltet das Blinken der mit SetFlash eingestellten Pulszeiten am Ausgang *Port* ab.

Deklaration: FlashOff (*Port* : Byte)

Erläuterung: Der Parameter *Port* kann die Werte [0 .. 5] annehmen.
Der Signalzustand des Ausgangs wird auf 0 geschaltet. Die mit SetFlash gesetzten Impulszeiten bleiben gesetzt.

Querverweis: SetFlash, FlashOn, BitOff, BitOn

Beispiel:
```
FlashOn(4);
SetClock(1,0);  {Clock 1 wird auf 0 gesetzt}
repeat until GetClock(1) > 30;
FlashOff(4);    {Das Blinken wird nach 30 s abgeschaltet}
```

7.5.3 Funktionen zum Lesen von Sensor- und Steuerzuständen

GetBit-Funktion

Funktion: liefert den Signalzustand des Ausgangs *Port*.

Deklaration: GetBit (*Port* : Byte)

Ergebnistyp: Byte

Erläuterung: Die Funktion gibt den Wert '0' zurück, wenn der Ausgang *Port* auf 0 geschaltet ist, und '1', wenn am Ausgang ein Signal anliegt. Der Parameter *Port* kann die Werte [0 .. 5] annehmen.

Querverweis: BitOn, BitOff, Sensor

Beispiel:
```
if (GetBit(0)=1) and (GetBit(1)=0)
then writeln ('Rechtslauf von Motor A');
```

Sensor-Funktion

Funktion: liefert den Signalzustand des Eingangs *Port*.

Deklaration: Sensor (*Port* : Byte)

Ergebnistyp: Byte

Erläuterung: Für den Parameter *Port* sind die Werte [6,7] gültig. Andere Werte erzeugen beim Programmlauf eine Fehlermeldung.

Die Funktion Sensor gibt den Wert '0' zurück, wenn an *Port* die grüne Signallampe nicht leuchtet. Im aktiven Zustand, wenn z. B. die Taste des LEGO Tastensensors gedrückt ist, ist das Ergebnis der Abfrage bei leuchtender Signallampe eine '1'.

Querverweis: GetBit

Beispiel: repeat until Sensor(6)=1;

GetPower-Funktion

Funktion: liefert den am Ausgang *Port* eingestellten Leistungspegel.

Deklaration: GetPower (*Port* : Byte)

Ergebnistyp: Byte

Erläuterung: Für den Parameter *Port* sind die Werte [0 .. 5, A, B, C, a, b, c] gültig.

Die Funktion gibt den an *Port* eingestellten Leistungspegel [0 .. 8] zurück. Wird ein Motorausgang abgefragt, wird bei unterschiedlichen Leistungspegelwerten an dem Portpaar nur der Leistungspegel ausgegeben, der beim Ausgang mit der niederen Nummer eingestellt ist.

Querverweis: SetPower

Beispiel:
```
if GetPower(A) < 8
then SetPower(A,GetPower(A)+1);
{Leistungspegel wird um 1 bis max. Wert hochgefahren}
```

Flashing-Funktion

Funktion: zeigt, ob an *Port* mit FlashOn das Blinken eingeschaltet ist.

Deklaration: Flashing (*Port* : Byte)

Ergebnistyp: Boolean

Erläuterung: Für den Parameter *Port* sind die Werte [0 .. 5] gültig.

Die Funktion Flashing gibt als Ergebnis den Wert 'true' zurück, wenn der Ausgang *Port* mit FlashOn auf 'blinken' geschaltet ist. Andernfalls wird der Wert 'false' zurückgegeben.

Querverweis: SetFlash, FlashOn, FlashOff

Beispiel: if Flashing(0) then FlashOff(0);
{Blinken wird ausgeschaltet}

7.5.4 Steuerung der Zähler

SetCounter-Prozedur

Funktion: stellt den Zähler *Port* auf den Zählerstand *Wert* ein.

Deklaration: SetCounter (*Port* : Byte; *Wert* : Longint)

Erläuterung: Für den Parameter *Port* sind Werte aus [6, 7, A, B, a, b] gültig. Mit dem Parameter *Wert* wird der Zähler auf seinen Startwert gesetzt.

Der LEGO TC Controller läßt die Steuerung von vier Zählern über die beiden Eingangskanäle 6 und 7 zu. Die beiden Zähler 6 und A bzw. a reagieren auf Signalflanken am Eingangskanal 6, während Signalflanken am Eingang 7 auf die beiden Zähler 7 und B bzw. b wirken.

Die Zähler 6 und 7 werden von dem mit SetCounter gesetzten *Wert* durch jede fallende Flanke hochgezählt. Eine fallende Flanke bildet der Übergang vom aktiven Sensor-

zustand in den Ruhezustand. Das Zurücksetzen der Zähler erfolgt mit dem Parameter *Wert* . Die Zähler 6 und 7 sind sofort mit dem Laden des LEGO TC Controllers in den Arbeitsspeicher des Computers aktiv, d. h. sie werden auch ohne Programmlauf mit jeder fallenden Signalflanke hochgezählt.

Die beiden Zähler A und B werden erst durch den Aufruf der Motor-Prozedur mit dem jeweiligen *Port* aktiviert. Dies kann auch in der Direktsteuerung vorgenommen werden. Beide Zähler können sowohl als Vorwärts- wie auch als Rückwärts-Zähler verwendet werden. Die Zählrichtung beim einzelnen Zähler ist abhängig vom gerade aktuellen oder zuletzt aktiven *Mode* rechts oder links; ist oder war zuletzt Rechtslauf eingeschaltet, zählt der Zähler vorwärts; Linkslauf veranlaßt den Zähler zum Rückwärtszählen. Anders als bei den Zählern 6 und 7 ändert bei den Zählern A und B jede steigende **und** jede fallende Signalflanke den zugehörigen Zählerstand.

Ein Deaktivieren eines einmal mit der Motor-Prozedur eingeschalteten Zählers A oder B ist selbst durch die ResetController-Prozedur nicht möglich. Der TC Controller muß dazu von der aktivierten Direktsteuerung aus mit **<Alt>-<x>** aus dem Arbeitsspeicher entfernt und wieder neu geladen werden.

Querverweis: GetCounter, CountFor, Frequency, Motor, ResetController

GetCounter-Funktion

Funktion: gibt den Zählerstand zurück.

Deklaration: GetCounter (*Port* : Byte)

Ergebnistyp: Longint

Erläuterung: Für den Parameter *Port* sind Werte aus [6, 7, A, B, a, b] gültig.

Mit dem Funktionsaufruf GetCounter werden die Zähler abgefragt und ausgelesen. Die Zähler 6 und 7 sind Vorwärtszähler, die mit jeder fallenden Signalflanke hochgezählt werden. Die Zähler A und B sind Vorwärts- oder Rückwärtszähler, abhängig je vom Zustand der Motorausgänge A und B. Nach er Anweisung Motor(A,rechts) zählt Zähler A vorwärts, nach Motor(A,links) rückwärts. Steigende wie fallende Flanken ändern den Zählerstand.

Vergleichen Sie auch die Erläuterung zur SetCounter-Prozedur.

Querverweis: SetCounter, Frequency, CountFor, Motor, ResetController

Beispiel:
```
SetCounter(6,0);
SetCounter(A,0);
Motor(A,rechts);
Wait(5);
writeln('Zählerstand 6: ', GetCounter(6));
writeln('Zählerstand A: ', GetCounter(A));
{Diese Anweisungssequenz zeigt das unterschiedliche
Zählverhalten der Zähler 6 und A.}
```

CountFor-Prozedur

Funktion: Zählschleife mit einstellbarem Grenzwert.

Deklaration: CountFor (*Port* : Byte; *Wert* : Longint))

Erläuterung: Für den Parameter Port sind Werte aus [6, 7, A, B, a, b] gültig.

Die Prozedur CountFor ist die Kurzform der mit dem TC Controller häufig benötigten Anweisungssequenz

Wert := .. ;
repeat until *Wert* = GetCounter(*Port*);

Querverweis: SetCounter, GetCounter, TimeOut

Beispiel:
```
uses LTCC;
begin
  SetPower(B,2);
  SetCounter(B,100);
  Motor(B,links);        {Rückwärtszähler}
  CountFor(B,-300);
  writeln('Ende');
  Motor(B,halten);
end.
```

{Der Motor wird mit SetPower auf einen niedrigen Leistungspegel eingestellt. Die Zählscheibe, die direkt auf der Motorachse steckt, rotiert mit der großen Teilung vor dem Optosensor. Der Motor wird dann nach 50 Zählscheibenumdrehungen - pro Umdrehung 8 Signalflanken - abgeschaltet.}

Frequency-Funktion

Funktion: liefert die Frequenz in der Einheit Hz.

Deklaration: Frequency (*Port* : Byte)

Ergebnistyp: Integer

Erläuterung: Für den Parameter *Port* sind die Werte 6 und 7 gültig.

Im 1/100stel Sekundentakt wird die Frequenz ausgelesen. Die Funktion berechnet aus den registrierten steigenden und fallenden Flanken die Zahl der Impulse je Sekunde und gibt das Ergebnis als Funktionswert, die Frequenz in Hz (=Hertz), zurück.

Querverweis: SetCounter, GetCounter

Beispiel:
```
SetPower(0,3);
SetFlash(0,0.1,0.1);
FlashOn(0);
Wait(3);
writeln(Frequency(6));
```
{Ein LEGO Leuchtbaustein wird mit der Flash-Prozedur bei verminderter Leistung ein- und ausgeschaltet. Das Licht wird über einen Optosensor am Eingangsport 6 detektiert.}

7.5.5 Steuerung der Uhren und Timer

SetClock-Prozedur

Funktion: stellt die Uhr *Nr* 1 oder die Uhr *Nr* 2.

Deklaration: GetClock (*Nr* : Byte; *Wert* : Real)

Erläuterung: Für den Parameter *Nr* sind Werte aus [1, 2] gültig. Andere Werte ergeben beim Programmlauf eine Fehlermeldung. Mit dem Parameter *Wert* wird die Stoppuhr auf ihren Startwert gesetzt: Der Ganzzahlanteil der Zahl bestimmt die Sekunden, die ersten beiden Stellen des Nachkommaanteils die 1/100stel Sekunden.

Mit der Prozedur SetClock werden zwei Stoppuhren gesteuert. Die Uhren zählen im Takt von 0,01 Sekunden vorwärts.

Im Gegensatz zur Prozedur SetClock werden mit der Prozedur SetTimer rückwärtszählende Stoppuhren gestartet.

Querverweis: SetTimer, GetClock

Beispiel:
```
SetClock(1,0);
Motor(A,links);
if (GetClock(1)>12.34) then Motor(A,wechseln);
if (GetClock(1)>25) then Motor(A,halten);
```

SetTimer-Prozedur

Funktion: stellt den Timer *Nr* 1 oder den Timer *Nr* 2.

Deklaration: SetTimer (*Nr* : Byte; *Wert* : Real)

Erläuterung: Für den Parameter *Nr* sind Werte aus [1, 2] gültig. Andere Parameterwerte ergeben beim Programmlauf eine Fehlermeldung. Mit dem Parameter *Wert* wird die Stoppuhr auf ihren Startwert gesetzt: Der Ganzzahlanteil der Zahl bestimmt die Sekunden, die ersten beiden Stellen des Nachkommaanteils die 1/100stel Sekunden.

Mit der Prozedur SetTimer werden zwei Stoppuhren gesteuert. Die Uhren zählen rückwärts im Takt von 0,01 Sekunden.

Erreicht ein Timer nach Ablauf der gesetzten Startwerts den Wert '0', dann hält die Stoppuhr an.

Querverweis: SetClock, GetTimer, TimeOut

Beispiel:
```
SetTimer(2,20.5);
BitOn(0);
repeat until GetTimer(2)=0;
BitOff(0);
```

GetClock-Funktion

Funktion: liest die Uhr 1 oder die Uhr 2.

Deklaration: GetClock (*Nr* : Byte)

Ergebnistyp: Real

Erläuterung: Für den Parameter *Nr* sind Werte aus [1, 2] gültig. Andere Parameterwerte ergeben beim Programmlauf eine Fehlermeldung.

Mit der Funktion GetClock wird die Uhrzeit in Sekunden mit einer Auflösung von 0,01 Sekunden ausgelesen.

Querverweis: SetClock, GetTimer

Beispiel:
```
SetClock(1,0);
repeat BitOn(0) until GetClock(1)>60;
BitOff(0);
repeat
  write(#7);
  Wait(0.1)
until GetClock>120;
```

GetTimer-Funktion

Funktion: liest die Timer *Nr* 1 oder *Nr* 2.

Deklaration: GetTimer (*Nr* : Byte)

Ergebnistyp: Real

Erläuterung: Für den Parameter *Nr* sind Werte aus [1, 2] gültig. Andere Parameterwerte ergeben beim Programmlauf eine Fehlermeldung.

Mit der Funktion GetTimer wird die Uhrzeit in Sekunden mit einer Auflösung von 0,01 Sekunden ausgelesen. Nach Ablauf der mit *Wert* in SetTimer gesetzten Sekunden bleibt die Uhr stehen und gibt den Wert '0' zurück.

Querverweis: SetTimer, SetClock, TimeOut

Beispiel:
```
SetTimer(1,30);
Motor(A,rechts);
if GetTimer(1)<15 then Motor(A,links);
repeat until GetTimer(1)=0;
Motor(A,halten);
```

TimeOut-Funktion

Funktion: zeigt an, ob der Timer auf dem Wert '0' angelangt ist.

Deklaration: TimeOut (*Nr* : Byte)

Ergebnistyp: Boolean

Erläuterung: Für den Parameter *Nr* sind Werte aus [1, 2] gültig.

Mit der Funktion TimeOut wird gestestet, ob beim Timer die gesetzte Laufzeit bereits abgelaufen ist und er den Wert '0' erreicht hat.

Querverweis: SetTimer, GetTimer, GetClock

Beispiel:
```
SetTimer(1,11.11);
Motor(A,links);
repeat write(#7) until TimeOut(1);
Motor(A,halten);
```

Wait-Prozedur

Funktion: stellt eine Wartezeit in Sekunden ein.

Deklaration: Wait (*Zeit* : Real)

Erläuterung: Mit der Wait-Prozedur wird die weitere Programmausführung um die mit dem Parameter *Zeit* übergebene Anzahl von Sekunden verzögert. Die Auflösung beträgt 0,01 s.

Querverweis: Delay(*ms* :Word) im Unit Crt, die um ms Millisekunden den Programmablauf verzögert.

Beispiel: Siehe Lauflichtprogramm Seite 174.

7.5.6 Steuerung des Controllers

ResetController-Prozedur

Funktion: stellt den LEGO TC Controller auf seinen Startwert zurück.

Deklaration: ResetController

Erläuterung: Mit der Prozedur ResetController wird der LEGO TC Controller wieder auf die Startwerte eingestellt. Dies bedeutet:

- Alle Ausgänge werden auf 0 zurückgeschaltet.
- Der Leistungspegel *Level* wird für jeden Ausgang auf den höchsten Wert 8 gesetzt.
- Die Pulsdauern *Mark* und *Space* der Flash-Prozedur erhalten den Wert 0,5.
- Alle Zähler (6, 7, A, B) werden auf 0 gesetzt.

Die ResetController-Prozedur steht sinnvollerweise als erste Anweisung in einem Steuerungsprogramm, da beim Start eines Steuerungsprogramms die TC Controller Einstellungen nicht automatisch zurückgestellt werden.

Direct-Prozedur

Funktion: schaltet die Direktsteuerung ein.

Deklaration: Direct

Erläuterung: Durch die Prozedur Direct wird aus einem laufenden Programm die Direktsteuerung aufgerufen. Die weitere Abarbeitung des Programms wird dadurch unterbrochen. Erst nach der manuellen Ausblendung des Direktsteuerungsfensters mit **<Esc>** wird das Programm mit den alten und den in der Direktsteuerung manuell geänderten Einstellungen wiederaufgenommen.

Mit dem Direct-Befehl kann man ein Steuerungsprogramm testen und beispielsweise bei irgendwelchen Stationen die Zählerstände überprüfen.

Während eines Programmlaufs läßt sich die Direktsteuerung vom Anwender natürlich jederzeit auch mit der Tastenkombination **<Alt>-<i>** einschalten.

7.6 Programmbeispiele für den TC Controller

7.6.1 Frequenzmessung

Das Programm im Edit-Fenster der integrierten Entwicklungsumgebung von Turbo Pascal (Abb. 7.10) demonstriert die Möglichkeiten der Frequency-Funktion. Beim Programmlauf wurden die Segmentwechsel der Zählscheibe gemessen, die direkt auf die Achse des Motors A gesteckt war und sich vor der Öffnung eines Optosensors drehte. Der Optosensor war an den Eingangskanal 6 angeschlossen. Das Meßergebnis ist im Output-Fenster angezeigt. Dieses kann mit der Tastenfolge **<F6>**, **<Alt>-<F6>**, **<F6>** gegen das Watch-Fenster ausgetauscht und zu

dieser Größe aufgezogen werden; dadurch sind die letzten Bildschirmausgaben ständig sichtbar. Unmittelbar nach dem Programmlauf wurde auch noch das Fenster der Direktsteuerung eingeblendet.

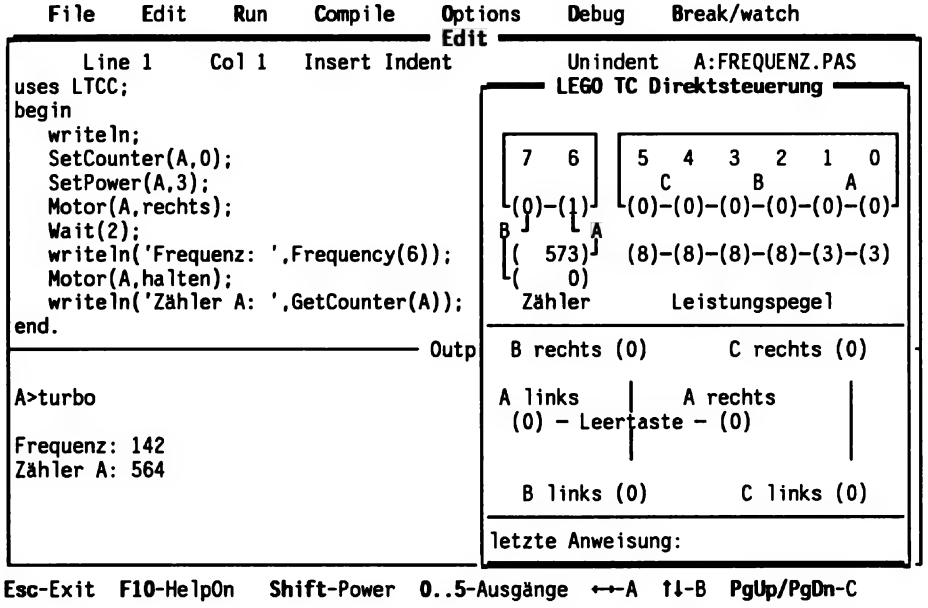


Abb. 7.10 Programmbeispiel für die Frequency-Funktion

Man sieht, daß der Zählerstand im Fenster der Direktsteuerung etwas größer ist, als bei der Messung mit dem Programm. Dies hängt damit zusammen, daß der Motor infolge seines Schwungs nach dem Abschalten noch einen gewissen Nachlauf hat; da der Zähler aktiviert ist, werden natürlich auch die Segmentwechsel mitgezählt, die nach dem Auslesen des Zählers durch das Programm noch stattfinden.

Für die Frequenz wurde vom Programm ein Wert von 142 Hz ausgegeben. Stimmt dieser Wert? Die Laufzeit des Motors betrug über die Wait-Prozedur 2 Sekunden. Vom Zähler A werden fallende und steigende Signalwechsel registriert. Zu einer vollen Periode gehören eine steigende und eine fallende Flanke. In 1 s fanden entsprechend 284 Signalwechsel statt. Da der Zähler 2 s lang zählte, können so mit dem

Frequenzwert 568 Signalflanken errechnet werden, was mit der Zählerausgabe recht gut übereinstimmt.

Der vom Programm ausgegebene Wert von 142 Hz zeigt, daß die Zähler in kürzeren Zeitintervallen als 0,01 s die Signalzustände an den Eingabeports abtasten und die momentanen Signalwerte dann wahrscheinlich in einem Ringpuffer ablegen, der in einer Sekunde vollgeschrieben wird. Dieser Ringpuffer wird dann entsprechend der Funktionsbeschreibung im Handbuch alle 0,01 s ausgewertet. Wäre dieser Auswertevorgang mit dem Abtastvorgang identisch, dann könnten gemäß dem Abtasttheorem der Informationstheorie gerade noch 50 Hz detektiert werden. Die von uns ermittelte Meßgrenze mit der Fequency-Funktion liegt aber bei etwa 150 Hz.

Aufgabe: Messen Sie die Abhängigkeit der Drehfrequenz der Motoren vom eingestellten Leistungspegel. Um alle Leistungspegel erfassen zu können, muß mit einer Zahnradübersetzung ins Langsame gearbeitet werden.

7.6.2 Datenübertragung

Die Datenfernübertragung wird bereits in naher Zukunft für jeden ein ganz wichtiges Anwendungsfeld des Computers sein. Die meisten Menschen werden dann nur mit Staunen und vielleicht auch mit Angst diesen Fähigkeiten der Computer gegenüberstehen. Mit dem LEGO TC Controller und der zugehörigen Hardware können und wollen wir an einem noch einfachen Beispiel die informationstechnischen Grundlagen eines jeden Datenverkehrs zwischen zwei Computern einsichtig erarbeiten und so mithelfen, wenigstens die Furcht vor dieser Technik abzubauen. Denn das Wissen um eine Sache ist die Voraussetzung, um sie letztlich zu beeinflussen und zu beherrschen.

Wir wollen zwei Computer zusammenschalten und jede Dateneingabe auf der Tastatur des einen auf dem Bildschirm des anderen anzeigen lassen. Jeder der beiden Computer kann und soll dann somit sowohl Sender wie Empfänger von Information sein. Als Informationskanal zwischen den beiden Computern dienen uns LEGO Interfaces und eine Lichtstrecke, die wir mit je einem Leuchtbaustein und einem Optosensor aufbauen. Die Lichtstrecke kann im Bereich von 0 bis ca. 0,5 m liegen. Der am

Interface des Computers A angeschlossene Leuchtbaustein strahlt das Signal einem am Interface des Computers B angeschlossenen Optosensors zu. Da die Information auf unserer Lichtstrecke aber nur in einer Richtung laufen kann, benötigen wir auch noch die Gegenrichtung: ein Kanal dient zum Senden und der andere zum Empfangen der Information. Das heißt, die Lichtstrecke zwischen Computer A und Computer B ist symmetrisch nochmals da. Am Interface des Computers A ist auch ein Optosensor angeschlossen, der durch das von der am Interface des Computers B angeschlossenen Lampe ausgehende Lichtsignal aktiviert wird. Beide Computer können somit Sender wie auch Empfänger sein.

Unsere direkte Computerkopplung erfolgt damit in einem Quasi-Duplexbetrieb über ein sogenanntes Nullmodem, weil der Datenaustausch in beiden Richtungen erfolgen kann und keine Umwandlung der digitalen Signale in analoge Signale wie bei Akustikkopplern erfolgt.

Die Zeichen werden als Binärmuster im ASCII-Code über die Lichtstrecke bitseriell übertragen. Der ASCII-Code (American Standard Code for Information Interchange) wurde für den digitalen Datenaustausch mit Fernschreibern noch vor der Computerära aufgestellt. In diesem Code kommt jedem Zeichen ein fester Zahlenwert zu: A wird beispielsweise durch die Zahl 65 verschlüsselt, was dann das Binärmuster 0100 0001 ergibt. Mit insgesamt 256 Signalmustern kann man das ganze Alphabet und daneben noch diverse Steuerzeichen, wie z. B. Wagenrücklauf, und grafische Zeichen verschlüsseln. Für jede 1 in einer solchen binären Signalfolge wird auf der Übertragungsstrecke ein Lichtimpuls gesendet, während bei einer 0 die Lampe dunkel bleibt.

Jeder Datenaustausch zwischen Computern erfordert in noch strengerem Maße als ein Gespräch oder eine Diskussion unter Menschen die Vereinbarung und die Beachtung von Verfahrensregeln. Gleichzeitiges Sprechen wie gleichzeitiges Senden führt niemals zu einem brauchbaren Ergebnis. Solche Verfahrensregeln bezeichnet man gemeinhin als Protokoll.

Wir verwenden in unserem Programm ein Handshaking für die Rollenverteilung, wann welcher Computer auf Sendung oder Empfang gehen kann. Mit diesem Handshaking-Signal wird dann auch in einem sog. XON-XOFF-Protokoll der Datenaustausch synchronisiert.

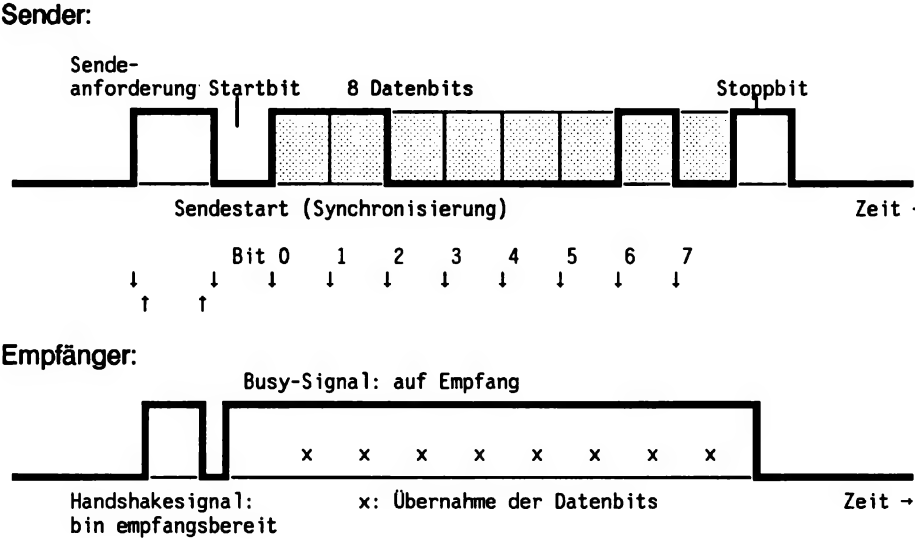


Abb. 7.11 Zeitdiagramm für das Sende-Empfangsprotokoll für das Zeichen C

Den zeitlichen Ablauf des Informationsflusses, wie er gemäß dem Protokoll in unserem Programm für die Übertragung des Zeichens C (ASCII-Code: 0100 0011) vonstatten geht, können wir Abb. 7.11 entnehmen.

Bevor jedoch der Datenaustausch nach diesem Protokoll vonstatten gehen kann, müssen in einem Deklarationsteil für beide Computer die notwendigen Vorgaben festgelegt werden. Dies erfolgt mit der Deklaration der verschiedenen Konstanten: Dies sind einmal die von der Übertragungsstrecke und der Beschaltung des LEGO Interfaces kommenden Vorgaben; an jedem Interface der beiden gekoppelten Computer ist am Ausgabeport 0 ein Leuchtbaustein und am Eingabeport 6 ein Optosensor angeschlossen. Dann sollen die zu übertragenden Zeichen alle in einem 8-Bit Binärmuster verschlüsselt sein. Weiter wird die Übertragungszeit für ein einzelnes Bit auf 0,3 s festgesetzt. Dies ergibt dann eine Übertragungsrate von etwas mehr als 3 Baud (=Bit/s). Die Übertragung eines Zeichens dauert damit insgesamt etwa 4 s.

Die Übertragungszeit darf nicht wesentlich kleiner gewählt werden, da durch die (Wärme-)Trägheit der Lampen der Infrarotdetektor im Optosensor sonst keinen Signalwechsel detektiert. Entscheidend ist hier insbesondere die Dauer der Ausphase der Leuchtbausteine. Die geringe Baudrate hat jedoch andererseits den Vorteil, daß die Übertragung der Information auf dem Informationskanal gut mitverfolgt werden kann. Man kann die Baudrate wesentlich erhöhen, wenn man als Lichtquelle Leuchtdioden, denen ein strombegrenzender Schutzwiderstand vorgeschaltet ist, anstelle der trägen Glühlämpchen der LEGO Leuchtbausteine verwendet.

Nachfolgend sehen Sie das kommentierte Programmlisting unseres Übertragungsprogramms, das im Anschluß daran für im Listinglesen Ungeübte auch noch ausführlich beschrieben ist. Um auch denjenigen Lesern das Ausprobieren dieses Übertragungsprogramms nicht vorzuenthalten, die nicht im Besitz des LEGO TC Controllers oder von Turbo Pascal sind, befindet sich neben dem Quellcode des Programms auch die lauffähige compilierte Programmdatei unter dem Namen SEMPF.EXE auf der Begleitdiskette zu diesem Buch.

```
program Senden_Empfangen;        {SEMPF.PAS}

uses CRT, LTCC;

const
  Port0    :Byte    = 0;        {Leuchtbaustein}
  Port6    :Byte    = 6;        {Optosensor}
  Port7    :Byte    = 7;        {Endesignal}
  clckNr   :Byte    = 2;        {Uhr und Timer für Protokoll}
  ein      :Byte    = 1;        {aktiv}
  aus      :Byte    = 0;        {inaktiv}
  maxBit   :Integer = 8;
  Taktzeit :Real    = 0.3;


procedure empfangen;
var
  time   :Real;
  i      :Integer;
  x      :Byte;

begin
  SetClock(clckNr,0);            {Clock zurücksetzen}
  BitOn(Port0);                  {Empfangsbereitschaft signalisieren}
  Wait(Taktzeit);
```

```
  BitOff(Port0);                {Handshake-Signal zurücksetzen}
  time := Taktzeit;             {Zeitspanne für 1 Bit}
  while (Sensor(Port6)=aus) and (GetClock(clckNr)<4*Taktzeit)
                do begin end; {auf Sendebeginn warten}
  if GetClock(clckNr)<4*Taktzeit then
  begin
    BitOn(Port0);               {Busy-Signal setzen}
    Wait(0.5*Taktzeit);         {Zeitoffset für Lesen}
    SetClock(clckNr,0);         {Synchronisierung: Clock für Empfang stellen}
    for i := 1 to maxBit do
    begin                       {Warteschleife für nächstes Bit}
      while GetClock(clckNr) < time do begin end;
      x := x shr 1;             {Bitmuster zurechtschieben}
      if Sensor(Port6) = aus then x := 128 + x;      {Bit einlesen}
      time := time + Taktzeit;  {Zeitmarke für nächstes Bit}
    end;
    write (Char(x));            {Zeichen auf Bildschirm ausgeben}
    BitOff(Port0);              {Busy-Signal zurücksetzen}
    time := time + Taktzeit;    {Zeit für Stoppbit setzen}
    while GetClock(clckNr) < time do begin end; {Stoppbitzeit abwarten}
  end
  else Wait(5*Taktzeit);
end;



procedure senden (Zeichen :Char);
var
  time  :Real;
  i     :Integer;
  x, y  :Byte;

begin
  if Zeichen = #7 then
    write (#7)                  {Piepston von Lautsprecher}
  else
  begin
    x := Byte(Zeichen);         {Typumwandlung}
    y := 0;                     {LOW-Signal für Startbit}
    time := (maxBit+2)*Taktzeit; {Zeitvorgabe setzen: maxBit Bit + }
    SetTimer(clckNr,time);            {+ Stoppbit + Startbit.}
    time := time - Taktzeit;    {Startzeitpunkt für Datenbit-Übertragung}
    for i := 0 to maxBit do     {8 Datenbits 'isolieren'}
    begin
      if y=0 then BitOff(Port0) else BitOn(Port0);  {Bit-Signalwert senden}
      y := x;           {Arbeitskopie von binärem Zeichenmuster herstellen}
      y := y shr 1;     {niederwertiges Bit durch Hinausschieben zuerst 0 }
      y := y shl 1;     {setzen und dann durch XOR-Vergleich mit altem Byte }
      y := x xor y;     {wiederherstellen während alle anderen Bits 0 werden.}
      x := x shr 1;     {Byte für nächsten Schleifendurchgang vorbereiten}
      while GetTimer(clckNr) > time do begin end;    {Taktintervall abwarten}
      time := time - Taktzeit;     {Zeit für nächstes Taktende setzen}
    end;
```

```
    BitOn(Port0);                  {Signalwert von Stoppbit setzen}
    repeat until TimeOut(clckNr);
    BitOff(Port0);                 {Stoppbit ausschalten}
    Wait((1+random(2))*Taktzeit);  {Verschnaufpause einlegen}
  end;
end;



function Taste :Char;
var flag :Boolean;
begin
  flag := true;
  Taste := #7;
  while flag and not TimeOut(clckNr) do
  begin
    if Sensor(Port6) = ein then
    begin
      Taste := ReadKey;       {Zeichen aus Tastaturpuffer lesen}
      flag := false;          {Marke für 'Zeichen gelesen' setzen}
    end;
  end;
end;



procedure sendeanforderung;
var
  flag   :Boolean;

begin
  if Sensor(Port6) = aus then empfangen
  else
  begin
    BitOn(Port0);                {Sendanforderung setzen}
    SetTimer(clckNr,4*Taktzeit); {max. Wartezeit vorgeben}
    flag := true;                {nur ein Zeichen senden}
    while (GetBit(Port0)=1) and flag do
    begin
      if Sensor(Port6) = aus then  {wenn Signal für Empfangsbereitschaft }
        begin                      { zurückgesetzt ist (Handshake)}
          flag := false;           {Flag für Schleifenende}
          SetTimer(clckNr,2*Taktzeit);  {max. Zeit für Zeichen lesen}
          senden(Taste);           {Zeichen lesen und senden}
        end
      else if TimeOut(clckNr) then
        begin                      {wenn Wartezeit verstrichen}
          BitOff(Port0);           {Zurücksetzen der Sendeanforderung}
          writeln('Empfänger meldet sich nicht');
        end;
    end;
  end;
end;                               {und Rückkehr zum Hauptprogramm}
```

```
{ Hauptprogramm }

begin
  ResetController;                {Controllerinitialisierung}
  while Sensor(Port7) = aus do
    begin
      if KeyPressed then sendeanforderung;
      if Sensor(Port6) = aus then empfangen;
    end;
  ResetController;
end.
```

Turbo Pascal Programm: SEMPF.PAS

Nach dem Start mit **<Ctrl>-<F9>** wartet das Programm im *Hauptprogramm* in einer while-Schleife darauf, bis es entweder durch Antippen einer Taste zum Senden eines Zeichens oder durch einen Signalwechsel am Eingangsport 6 zum Empfang eines Zeichen veranlaßt wird. Mit einem irgendwie aktivierten Eingabeport 7, der in der while-Bedingung abgefragt wird, kann das Programm auch insgesamt geordnet beendet werden.

Hat das Programm mit der Funktion *KeyPressed* einen Tastendruck wahrgenommen, dann meldet es durch Anschalten der Lampe am Ausgangsport 0 seinen Sendewunsch an. Vorher prüft es jedoch am Eingangsport 6 noch nach, ob zwischenzeitlich vielleicht vom anderen Computer schon eine Sendeanforderung abgeschickt worden ist. Damit bei nicht erwiderter Empfangsbereitschaft nicht ewig gewartet wird und um sicherzugehen, daß nur ein Zeichen gesendet wird, werden Vorgaben gesetzt, die dann in der nachfolgenden while-Schleife als Abbruch- bzw. Verzweigungskriterium abgefragt werden.

Meldet der Partnercomputer auf Eingangskanal 6 mit einem Lichtsignal seine Empfangsbereitschaft, dann erst wird die eigentliche Prozedur *senden* angesprungen. Dabei muß jedoch vorher noch mit der Funktion *Taste* das zu sendende Zeichen aus dem Tastaturpuffer abgeholt werden. In dieser Funktion wird die gedrückte Taste dann ausgelesen, wenn der Partnercomputer sein Bereitschaftssignal wieder zurückgesetzt hat. Sollte das Zeichenholen aber aus irgend einem Grund mit einem Mißerfolg enden, dann gibt der Computer wegen des Funktionsrückgabewertes von #7 wenigstens einen Ton von sich.

In der Prozedur *senden* wird nun nach dem Setzen der gesamten Sendezeit und dem Senden des sog. Startbits - mit diesem Zurücksetzen des Sendeanforderungssignals wird der Partnercomputer gleichzeitig synchronisiert -, in einem festen Takt das Binärmuster des zu sendenden Zeichens Bit für Bit auf den Informationskanal gegeben. Die bitserielle Übertragung beginnt dabei mit dem niederwertigsten Bit 0.

Nach dem 8. Bit wird als Stoppbit in jedem Fall die Signallampe nochmals für ein Taktintervall eingeschaltet, bevor dann mit dem Abschalten des Lichtsignals und einer kurzen weiteren Warteperiode das Programm wieder in die Warteschleife des Hauptprogramms zurückkehrt.

Während der eine Computer in die Senderrolle geschlüpft ist, hat sich der Partnercomputer auf das Signal der Sendeanforderung hin auf das Empfangen eingestellt. In der Prozedur *empfangen* wird zuerst Empfangsbereitschaft signalisiert, indem auf dem Nachrichtenkanal das Lichtsignal für ein Taktintervall eingeschaltet wird. Danach wartet der Computer für höchstens eine eingestellte Zeit (4 Taktzeiten) auf den Sendestart, der ihm über Eingangskanal 6 signalisiert wird. Das Signal dazu kommt jedoch im Normalfall unmittelbar nach dem Zurücksetzen des Handshake-Signals.

Nach dem Empfang der fallenden Signalflanke des Startbits setzt der Empfangscomputer seine Sendeleitung auf busy zum Zeichen dafür, daß er nun für die nächste Zeit beschäftigt ist und synchronisiert sich durch das Nullsetzen seiner Clock auf den Sendetakt des Senders. Immer in der Mitte eines Taktintervalls holt sich dann der Empfänger auf dem Informationskanal das gerade dort anliegende Signalbit ab und reiht die nacheinander eintreffenden Bits wieder zu dem Binärmuster des gesendeten Zeichens zusammen. Nach dem Empfang des 8. Bits wird das empfangene Binärmuster in das zugehörige Zeichen umgesetzt und auf dem Bildschirm ausgegeben. Schließlich wird das sog. Busy-Signal noch zurückgenommen und dann nach einem Stopptakt ebenfalls in die Hauptprogrammschleife zurückgekehrt.

Sollte das Handshaking nicht erfolgreich abgelaufen und die Wartezeit verstrichen sein, dann läßt der Empfangscomputer nochmals 5 Takte Wartezeit verstreichen, um zusammen mit seinem Partnercomputer wieder in einen gemeinsamen neuen Wartezustand zu gelangen. Dadurch ist

sichergestellt, daß die einmal außer Tritt gekommenen Computer sich wieder fangen können.

Aufgabe: Ändern Sie das Programm so ab, daß bei Simplexbetrieb, d. h. die Übertragung von Daten ist nur in einer Richtung möglich, die zu sendenden Zeichen auch auf dem Sendebildschirm zur Anzeige gebracht werden.

Aufgabe: Ändern Sie das Programm so ab, daß Sie die Sendedaten nicht von der Tastatur abholen, sondern von einer auf Diskette gespeicherten Datei einlesen.

Eine Fundgrube für weitere Steuerungsprogramme ist das Lehrerhandbuch zum LEGO TC Controller. Dort findet man eine ganze Reihe ausgeführter Beispiele mit den zugehörigen Modellbeschreibungen und auch mit didaktisch-methodischen Kommentaren.

Anhang

LEGO Lines Strukturanweisungen

In LEGO Lines stehen folgende Struturanweisungen zur Verfügung:

WIEDERHOLE - IMMER
Wiederholt den durch die beiden Anweisungsteile eingeklammerten Programmabschnitt fortwährend (Endlosschleife).

WIEDERHOLE n - ENDE WIEDER
Wiederholt den durch die beiden Anweisungsteile eingeklammerten Programmabschnitt genau **n**-mal.

WIEDERHOLE - BIS
Wiederholt den durch die beiden Anweisungsteile eingeklammerten Programmabschnitt, bis die in den Spalten 6 und 7 der BIS-Zeile angegebene Bitkombination an den Eingabekanälen des Interfaces anliegt.

WENN - ENDE WENN
Die durch die beiden Anweisungsteile eingeklammerte Programmsequenz wird nur dann abgearbeitet, wenn die in der WENN-Zeile angegebene Bitkombination an den Eingabekanälen 6 und 7 des Interfaces anliegt.

ZÄHLE n
Der Programmablauf wird solange gestoppt und die zuletzt eingeschaltete Bitkombination gehalten, bis am angegebenen Eingabekanal genau **n** Impulswechsel registriert wurden.

LEGO Lines Funktionstasten

Funktion	**IBM-PC's u. Komp.**	**Commodore 64 o. 128**
Zeile einfügen	<F3>	<f1>
Zeile löschen	<F4>	<f3>
Test	<F10>	<f5>
Run (Sichtlauf)	<F9>	<f7>
Run (Ablaufmodus)	<Shift>-<F9>	-
Stop	<F1>	<RUN-STOP>
Programm löschen	<Ctrl>-<F2>	<CTRL>-<f3>
Beenden	<Ctrl>-<F8>	<CTRL>-<f7>
Ausdrucken von Steuerungsprogrammen	<F6>	<CTRL>-<f1>
Diskettenzugriff	<F5>	<f2>
Direktmodus	<F7>	<f6>

Tastenkorrespondenzen
auf unterschiedlichen Tastaturen von IBM-PC's und Kompatiblen

<Strg>	<Ctrl>
<Einfg>	<Ins>
<Entf>	<Del>
<Pos1>	<Home>
<Ende>	<End>
<Bildauf>	<PgUp>
<Bildab>	<PgDn>
<Druck>	<PrtSc>
<Rollen>	<Scroll>
<⇑>	<Shift>

Literaturhinweise

Härtl, Alfred: Optoelektronik in der Praxis
Härtl-Verlag, ohne Ort, ohne Jahr.

Heimsoeth Software GmbH (Hrsg.): Turbo C Band I: Benutzerhandbuch
München, 1988

Heimsoeth Software GmbH (Hrsg.): Turbo C Band II: Referenzhandbuch
München, 1988

Heimsoeth Software GmbH (Hrsg.): Turbo Pascal 4.0, Band I:
Benutzerhandbuch. München, 1987

Heimsoeth Software GmbH (Hrsg.): Turbo Pascal 4.0, Band II:
Referenzhandbuch. München, 1987

Heimsoeth Software GmbH (Hrsg.): Turbo Pascal,
Addendum Version 5.0. München, 1988

Horn, Klaus: PC-Nutzung mit TURBO-PASCAL
Stuttgart: Teubner, 1989

Jansen, Jan Hendrik: Anwendung digitaler Bausteine
München: Franzis, 1985

LEGO Group: Lehrerhandbuch zu LEGO Technic Control I,
Software LEGO Lines: Commodore 64, 128
Hohenwestedt: LEGO GmbH, 1987

LEGO Group: Lehrerhandbuch zu LEGO Technic Control I,
Software LEGO Lines: IBM PC und kompatible Computer
Hohenwestedt: LEGO GmbH, 1988

LEGO Group: Softwarehandbuch zum LEGO TC Controller,
Computer: IBM PC und Kompatible
Hohenwestedt: LEGO GmbH, 1988

LEGO Group: Lehrerhandbuch zu LEGO Technic Control II,
Software: LEGO TC Controller
Hohenwestedt: LEGO GmbH, 1989

Mechelke, Günther: Einführung in die Analog- und Digitaltechnik
Köln-Porz: Stam, 1985

Schulé, Roland: Einführung in die Mikroprozessor-Anwendung
Köln: vgs, 1987

Schwarze/Holzgrefe: Computereinsatz beim Regeln und Steuern
Stuttgart: Metzler/Teubner, 1987

Stiller, Andreas: PC-Bausteine. Rund um den Timer
in: c't 4/1988, Seite 196ff.

UniComal Documentation Center: Comal Einführung
Rodding, 1988

UniComal Documentation Center: Comal Reference Manual
Rodding, 1988

Hinweise zu der Begleitdiskette

Dieser Band wird mit einer Begleitdiskette für IBM-PC's und Kompatible ausgeliefert. Die Diskette ist in die Unterverzeichnisse:

<BASIC>
<COMAL>
<LINESFIL>
<LTCC>
<TURBOC>

aufgeteilt und enthält die im Buch aufgeführten Programmdateien.

Die Nutzung der Diskette ist nur dem Käufer persönlich gestattet. Schulen erhalten bei schriftlicher Anfrage die Genehmigung zur Herstellung von Kopien für den Unterrichtseinsatz innerhalb der Schule.

Hinweise zum Video

Unter der Bestellnummer M/T 70277 ist beim Verlag J. B. Metzler, Kernerstraße 32, 7000 Stuttgart 1 das VHS-Video

Steuern und Regeln mit LEGO Lines

für DM 45,-- (bzw. DM 20,-- im Verleih für 20 Tage) erhältlich. Beim Bezug von LEGO Materialien durch den J. B. Metzler Verlag erhalten Sie die Gebühr rückvergütet.

Der erste Teil des Films zeigt die Konfiguration der Hardware und die Funktionsbausteine der Technic Control Sets. Es folgt eine schrittweise Einführung in LEGO Lines. Anschließend wird anhand von Unterrichtsmitschnitten aus dem Projekt "Sortieranlage" der konkrete Einsatz der LEGO Materialien im Unterricht gezeigt.

Sachverzeichnis

ComputerPraxis im Unterricht Fortsetzung

Die Metzler + Teubner Buch- und Diskettenreihe für die allgemeine und berufliche Lehrer- und Erwachsenenbildung

Neubeck: **Computereinsatz im Musikunterricht**
In Vorbereitung

Schwarze/Hamann: **Computereinsatz in der Meßtechnik**
197 Seiten. Buch mit Diskette (IBM+kompatible) DM 48,–

Schwarze/Holzgrefe: **Computereinsatz beim Regeln und Steuern**
204 Seiten. Buch mit Diskette (IBM+kompatible) DM 48,–

Steidl: **Computereinsatz im Chemieunterricht**
260 Seiten. Buch mit Diskette (IBM+kompatible) DM 48,–

Werner u. a.: **Schüler arbeiten mit dem Computer**
Materialien für die Sekundarstufe I
272 Seiten. Buch mit Diskette (IBM+kompatible) DM 48,–

Wespel: **Computereinsatz im Deutschunterricht**
142 Seiten. Buch mit Diskette (IBM+kompatible) DM 48,–

Preisänderungen vorbehalten

Computer Praxis im Unterricht

Die Metzler+Teubner Buch- und Diskettenreihe für die allgemeine und berufliche Lehrer- und Erwachsenenbildung

Best. Nr. M/T 70214